all rights reversed

HACKER

Μάιος 2014 # Τιμή τεύχους 4€

τεύχος 032 - #DropDropbox Edition

PATCHER CRACKER PACKER

Ταχύρυθμα μαθήματα program cracking [*]

Η τεχνική του port knocking

Ανάπτυξη εφαρμογών για το Firefox OS

Φτιάξτε το δικό σας cloud storage system

Stack overflow attacks για κοινούς θνητούς

[*] ...και αντίμετρα με τη βοήθεια των patchers

all rights reversed

HACKER

tefhos 32o

Περιεχόμενα

και οδηγίες χρήσης

56 Ανάπτυξη εφαρμογών για το Firefox OS [Μέρος 2]

Όπως είδαμε, η ανάπτυξη εφαρμογών για το Firefox OS είναι αρκετά εύκολη υπόθεση και δεν διαφέρει ιδιαίτερα από τη δημιουργία μιας σύγχρονης ιστοσελίδας. Πώς όμως η εφαρμογή μας μεταφέρεται σε μια συσκευή με το Firefox OS;

70 Αφήστε τους να κρυφακούνε

Κι αμπάρες βάλαμε στους servers μας, και σύρτες κι απ' όλα. Μέχρι και την πόρτα κρύψαμε, αξιοποιώντας την τεχνική του port knocking. Όμως οι παρανοϊκοί με την ασφάλεια δεν ικανοποιούνται ποτέ. Και καλά κάνουν, δηλαδή.

online deltaCast s02e03 | DropDropbox - Part 1/2

http://deltahacker.gr/deltacast-s02e03

online deltaCast s02e04 | DropDropbox - Part 2/2

http://deltahacker.gr/deltacast-s02e04

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Γιώργος Αραβίδης, Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Χρήστος Θεμελής, Μανώλης Κιαγιάς, Ιωάννης Κονιάρης, Πέτρος Κυλαδίτης, Γιώργος Μελέκος, Νίκος Μουρατίδης, Νικόλαος Σηφάκης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το ηλεκτρονικό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, κάθε μήνα. Η Parabing Creations έχει την έδρα της στη Μαιώτιδος 3, 55133, Καλαμαριά. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/subscriptions και οι ίδιες οι παραγγελίες γίνονται από το http://deltahacker.gr/pdf12order. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.


delta HACKER
Στη 3η χρονιά και συνεχίζουμε ακάθεκτοι!
Μηνιαίο περιοδικό deltaHacker, http://deltahacker.gr
Ethical Hacking / Δίκτυα / Ασφάλεια / Προγραμματισμός / Ηλεκτρονικά
twitter.com/deltaHacker
facebook.com/deltaHacker

Χωρίς γιρλάντες, χωρίς καραμούζες

Αγαπητές κι αγαπητοί,

- Αυτή τη στιγμή που σας γράφω, εδώ στη Θεσσαλονίκη έχουμε σε εξέλιξη ένα πλήρες demo του φθινοπώρου. Κατά πώς φαίνεται, μάλιστα, θα διαρκέσει όλη την ημέρα.
- Τις προάλλες έμαθα ότι ο Θείος Ακάκιος πάλι έφυγε για το Λονδίνο, όπου επισκέπτεται τον ανηψιό. Τον ζηλεύω λίγο που έχει ευκαιρίες για ταξιδάκια προς τα εκεί, αλλά τις μέρες αυτές όχι και τόσο. Είναι που ο τοπικός καιρός θυμίζει κάπως Λονδίνο, κι όπως πιθανώς θα γνωρίζετε εμένα πολύ μ' αρέσουν η δροσιά και η μουντάδα (του καιρού).
- Εντάξει, το καλοκαίρι δεν κινδυνεύει και είμαι σίγουρος ότι πολύ σύντομα θα μας επιτεθεί για τα καλά και τίποτε δεν θα μπορέσει ν' ανακόψει την επέλασή του. Αυτή η βεβαιότητα όμως καθόλου δεν περιορίζει τη ζωηράδα που με 'χει πιάσει, λόγω *και* αυτού του φθινοπωρινού demo.
- Σκεφτείτε το: Ζέστη δεν κάνει, ο ήλιος δεν σε στραβώνει ούτε σε ζαλίζει όταν βγαίνεις έξω για περίπατο, η μέρα είναι μεγάλη. Ό,τι πρέπει, δηλαδή, για να αφοσιώνεσαι άνετα κι ωραία στις εργασίες και στα πρότζεκτ σου – να κάνεις και τα διαλείμματά σου.
- Κοιτάξτε, ιδιαίτερα εργασιομανής δεν είμαι. Έχουμε όμως βάλει μπρος κάποιες νέες δουλειές, είμαστε ενθουσιασμένοι που τα σχέδιά μας παίρνουν μορφή και γενικά αυτό το καλοκαίρι αναμένεται να 'χουμε major πήξιμο.
- Ορίστε, το είπα χωρίς περιστροφές: Θα έχουμε major πήξιμο.
- Το νου σας, ο πληθυντικός που χρησιμοποιώ δεν είναι της ευγένειας. Είμαστε πράγματι πάνω από ένας, όσοι θα πήξουμε το καλοκαίρι τούτο. Ευτυχώς, κανείς μας δεν παραπονιέται γι' αυτό. Ίσα ίσα που το ακριβώς αντίθετο συμβαίνει...
- Να, ο κος Αρχισυντάκτης (TM) για παράδειγμα, πρακτικά έχει τελειώσει με τη σχεδίαση του site της αγγλικής έκδοσης του deltaHacker. Μπορείτε να δείτε τη δουλειά του στο `http://parabing.com`. Τη στιγμή που γράφονται οι γραμμές τούτες στο Parabing αληθινό περιεχόμενο δεν υπάρχει, ενώ κάποια οπτικά στοιχεία δεν είναι τελικά. Μέσα στις επόμενες εβδομάδες ορισμένα πράγματα θ' αλλάξουν και βεβαίως θα αναρτάται αληθινό περιεχόμενο, αλλά προς το παρόν άλλο θέλω να σας πω...
- Αρχικά, που λέτε, από το Parabing θα προσφέρουμε μόνο δωρεάν περιεχόμενο, υπό τη μορφή κειμένου και βίντεο. Στα αγγλικά βέβαια – κι όχι κάτι που δεν δημοσιεύεται στο deltaHacker ή δεν παρουσι-

άζεται στα επεισόδια του deltaCast. Μετά από έναν ή δύο μήνες, το Parabing θα παρέχει έναν συνδυασμό δωρεάν κι επί πληρωμή περιεχομένου. Οι απλοί επισκέπτες θα έχουν ευκαιρία να βλέπουν και ν' αξιολογούν τη δουλειά μας, ενώ όσοι αγοράζουν (ετήσια) συνδρομή θα αποκτούν πρόσβαση σε όλο το περιεχόμενο ανεξαιρέτως.

- Σε κάθε περίπτωση, η προσοχή και η έμφαση δίνονται *ακριβώς* σ' αυτό: στο περιεχόμενο.
- Χρειαζόμασταν, λοιπόν, ένα theme για το site που να καταδεικνύει τη φιλοσοφία μας όσο το δυνατόν καλύτερα. Το περιεχόμενο είναι Βασιλιάς, όπως λένε, οπότε όλα τα στολίδια, οι γιρλάντες και οι καραμούζες, δεν μπορούν παρά να περιττεύουν.
- Μη σας πω κιόλας ότι σε αρκετές περιπτώσεις όλα αυτά τα παραφερνάλια υπάρχουν σκόπιμα, με στόχο να αποπροσανατολίζουν τον επισκέπτη μπας και δεν αντιληφθεί ότι υπάρχει (σοβαρό) έλλειμμα περιεχομένου.
- Κοιτάξαμε λοιπόν γύρω γύρω για WordPress themes που να μας κάνουν και, πράγματι, βρήκαμε κάποια υποψήφια. Κανένα όμως δεν φώναζε "εγώ είμαι για 'σας". Και κάπως έτσι, που λέτε, πήρε ο κος Αρχισυντάκτης (TM) τη μεγάλη απόφαση ν' αναπτύξει το δικό του theme ξεκινώντας από το μηδέν.
- Αρκετό καιρό μετά –είναι και το αγαπημένο μας deltaHacker που έχουμε και θέλουμε να φροντίζουμε–, το αποτέλεσμα της δουλειά του είναι ένα καθαρό theme για το Parabing, με δύο επιθυμητά χαρακτηριστικά: α) responsive, ώστε το site να είναι κατάλληλο για κάθε συσκευή που έχει ο επισκέπτης, β) καθαρό, ώστε ο αναγνώστης/θεατής να αφοσιώνεται ευκολότερα στο ίδιο το περιεχόμενο και η προσοχή του να μην αποσπάται από δευτερεύοντα (και τριτεύοντα) στοιχεία.
- Υπήρχαν πολλές ενδιαφέρουσες προκλήσεις και προβλήματα που συνάντησε κι αντιμετώπισε ο κος Αρχισυντάκτης (TM). Μου τα έλεγε και μου τα εξηγούσε, όμως να σας πω την αλήθεια δεν είμαι ο καταλληλότερος άνθρωπος για να σας τα μεταφέρω. Ίσως πρέπει να κανονίσουμε ένα live stream με co-host τον Παναγιώτη[1], να μας τα πει καλύτερα εκείνος και γενικά να τα πούμε με την άνεσή μας.
- Ναι, το έχουμε παραμελήσει το deltaStream, το ξέρω. Είναι που μας έχουν πέσει πολλά και μαζεμένα τώρα τελευταία – κι εκτός από το deltaHacker και το Parabing έχουμε βεβαίως και τα deltaCast episodes.
- Ξέρετε κάτι; Εδώ και βδομάδες προσπαθώ να μη λέω και να μη γράφω "deltaCast episodes", "επεισόδια του deltaCast", "το deltaCast" κ.ο.κ.
- Είναι που κάποιοι νέοι φίλοι και συνδρομητές με ρωτούν τι είναι το deltaCast. Ξεκινάω λοιπόν να εξηγώ ότι είναι το video podcast του

1. Also Known As "Ο κύριος Αρχισυντάκτης (TM)".

περιοδικού deltaHacker, αμέσως όμως με διακόπτουν και με ρωτούν τι είναι αυτό το "podcast". Με το δίκιο τους. Ποιος χρησιμοποιεί σήμερα τον όρο "podcast", εκτός ίσως από εκείνους που έζησαν την επανάσταση του podcasting; Εντάξει, δεν έχουν περάσει και πολλά χρόνια από τότε που ξεκίνησαν –πρώτα τα audio και μετά τα video– podcasts, όμως για το ρυθμό των εξελίξεων στο Internet είναι σαν να έχουν περάσει πολλές δεκαετίες.

- Γι' αυτό λοιπόν κι εγώ προσπαθώ να μιλάω για video tutorials, αφού τέτοια είναι τα deltaCast episodes.
- Εσείς παρακολουθήσατε το πλέον πρόσφατο, διπλό επεισόδιο;
- Ώπα, ξανά.
- Παρακολουθήσατε τα δύο πρόσφατα video tutorials; Αφορούν στην αντικατάσταση του παραδοσιακού cloud storage system –είπε κανείς Dropbox;– με ένα δικό μας, το οποίο αφενός μας δίνει πίσω τον έλεγχο των δεδομένων μας, αφετέρου διατηρεί τις ευκολίες που παρέχει το cloud.
- Προσέξτε: Για να παρακολουθείτε τα video tutorials του deltaHacker, χρειαζόσαστε ενεργή συνδρομή στο περιοδικό. Κι αν δεν έχετε, αυτές τις μέρες έχουμε ξεκινήσει τις Lifetime subscriptions τις οποίες πραγματικά *δεν* θέλετε να αγνοήσετε – αν μη τι άλλο εξαιτίας της μεγάλης έκπτωσης που προσφέρουμε για λίγες μέρες (`http://deltahacker.gr/?p=11760`).
- (Αν δεν προλάβετε την έκπτωση, στείλτε μας ένα email στο talk2us@deltahacker.gr και θα δούμε τι μπορούμε να κάνουμε.)
- Και μετά απ' αυτή τη χωρίς ντροπή διαφήμιση, ας επιστρέψω στο πήξιμο.

Καλά να περνάτε,

Ο Προκομμένος Σας (TM)



ονέλ μέλ

Αγαπητοί μου,

Ας αρχίσουμε από τα σημαντικά.

Είναι Τετάρτη, 28 του μηνού, και είμαι με τον ανηψιό στη Βιβλιοθήκη. Είμαστε καθισμένοι σε διπλανά τραπέζια του Humanities I. Πρόκειται για το αναγνωστήριο που προτιμάει ο ανηψιός, γιατί έχει ψηλό ταβάνι και αρκετό φυσικό φως.

Κατεβήκαμε μαζί, αυτός γιατί είχε ζητήσει κάτι βιβλία να τα κοιτάξει, κι εγώ γιατί δεν είχα τι να κάνω στο σπίτι. Εφερα φυσικά και τη γραφομηχανή μαζί μου, αλλά αναγκάστηκα να την αφήσω στο βεστιάριο γιατί δεν θα με άφηναν με τίποτα να τη βάλω στο αναγνωστήριο.

Από προχτές Δευτέρα που έφτασα στο Λονδίνο, επομένη των εκλογών, ο καιρός είναι μίζερος: Ψιλόβροχο, ψύχρα και μουντάδα. Λες και το κάνει επίτηδες. Αλλά εντάξει. Αναμενόμενο. Τι άλλο να περιμένει κανείς Μάη μήνα, στη Γηραιά Αλβιώνα; Θα την ξέρετε δα την παροιμία για το Μάη στον καταραμένο τόπο, έτσι;

Είμαστε που λέτε στον πρώτο όροφο, της Βρετανικής Βιβλιοθήκης, στο Humanities I, και κοιτάζω τα περίεργα όντα γύρω μας. Όλοι σκυμμένοι στα βιβλία και τους κομπιούτορές τους. Χαρτιά απλωμένα, σημειώσεις, κόντρα σημειώσεις, ντοσιέ, κόντρα ντοσιέ, φωτοτυπίες... Χαμός.

Εγώ έχω κάτι σελίδες Α4 που μου έδωσε ο ανηψιός, και σας γράφω με το μολύβι που αγόρασα από το σουβενιράδικο της βιβλιοθήκης. Θα το μεταγράψω στη γραφομηχανή αργότερα, που θα γυρίσουμε σπίτι.

Τέλος πάντων, η βιβλιοθήκη έχει πλάκα και χαίρομαι που έκανα τον κόπο να γραφτώ μέλος πέρσυ, τότε που είχαμε ξαναέρθει στο Λονδίνο. Το είχαμε μαϊμουδέψει φυσικά λίγο, καθώς είχα δώσει για διεύθυνση μόνιμης κατοικίας τη διεύθυνση του ανηψιού, αλλά εντάξει.

Λοιπόν, μου έχουν μείνει κάποια πράγματα να σας πω, εκκρεμότητες, από τον περασμένο μήνα.

Κυρίως για τον Χώκινγκ.

Το συζητούσαμε τις προάλλες με τον ανηψιό. Κυριακή των εκλογών, κι εγώ ήμουν ακόμα στην Αθήνα. Μιλάγαμε οικογενειακώς στο σκάιπ, περιμένοντας τα έξιτ πολλ.

"Και πώς είσαι σίγουρος ότι ο Χώκινγκ κάνει το λάθος που του προσάπτεις;", με ρωτάει σε κάποια φάση ο ανηψιός. "Γιατί λες ότι πήγε να μιλήσει για κανονικότητες και του βγήκε να πει για προθετικότητες; Πού το ξέρεις; Επιστήμονας είναι ο άνθρωπος, η κοσμολογία τον ενδιαφέρει, όχι οι προθετικότητες. Για κανονικότητες μίλαγε. Νομίζω ότι στο συγκεκριμένο θέμα το λάθος το έχεις κάνει εσύ, και όχι αυτός."

"Δε νομίζω", του λέω εγώ. "Στον πρόλογο το γράφει καθαρά. Λέει ότι με το βιβλίο του θα επιχειρήσει να απαντήσει σε πανάρχαια και πανανθρώπινα ερωτήματα, όπως για παράδειγμα, 'Γιατί υπάρχει κάτι παρά τίποτα;', 'Γιατί

υπάρχουμε;' και πάει λέγοντας. Τονίζει, μάλιστα, ότι αυτά τα ερωτήματα ήταν από πάντα ερωτήματα της φιλοσοφίας, 'αλλά η φιλοσοφία έχει πεθάνει'."

"Ε και;"

"Τι ε και; Δεν το βλέπεις; Αντίθετα με αυτό που θέλει να πιστεύει ο Χώκινγκ, όταν οι άνθρωποι ρωτάνε 'γιατί υπάρχουμε;' δεν το κάνουν γιατί τους ενδιαφέρει η κοσμολογία αλλά γιατί τους πιάνουν τα υπαρξιακά τους. Θέλουν να καταλάβουν το νόημα της ζωής. Κι από αυτό που καταλαβαίνουν εξαρτώνται πολλά. Ο ένας πάει και γίνεται ερημίτης. Ο άλλος αυτοανατινάζεται στο παζάρι της Βαγδάτης. Ο ένας γίνεται μεγάλος φιλάνθρωπος. Ο άλλος αποφασίσει να οργανώσει την 11η Σεπτεμβρίου. Οι αντάρτες της Μπόκο Χαράμ πάνε στα σχολεία και αρπάζουν τις μαθήτριες για να τις αναγκάσουν να ασπαστούν το Ισλάμ. Και πάει λέγοντας."

"Τι σχέση έχει αυτό με αυτά που λέει ο Χώκινγκ;"

"Καμία. Αλλά αυτό ακριβώς εννοώ. Ότι ο Χώκινγκ μπαίνει σε άγνωστα νερά χωρίς να το πάρει είδηση. Επειδή πιστεύει πως κατάλαβε το πώς λειτουργεί η διαδικασία της 'αυθόρμητης δημιουργίας', νομίζει ότι μπορεί να απαντήσει στο ερώτημα 'γιατί υπάρχουμε' και 'γιατί είμαστε εδώ'. Διακηρύσσει το θάνατο της φιλοσοφίας, χωρίς να βλέπει ότι κάνει a little step too far."

Μας διέκοψε με έξαψη η μητέρα της Εξαδέλφης. "Ελάτε, ελάτε, αρχίζουνε τα έξιτ πολλ!"

"Αμάν καλέ μαμά, πώς κάνετε έτσι με τα έξιτ πολλ;" διαμαρτυρήθηκε η Εξαδέλφη, σέρνοντας την καρέκλα της πιο κοντά στην τηλεόραση. "Άμα είναι να μην παιδευόμαστε με τις εκλογές, να κάνουμε ένα έξιτ πολλ που είναι και φτηνότερο και να τελειώνουμε..."

"Άσε τις εξυπνάδες" αντέτεινε η μητέρα της Εξαδέλφης που είναι Ινδουίστρια. "Τα έξιτ πολλ δίνουν τον παλμό της υπόλοιπης βραδιάς των αποτελεσμάτων, και γι αυτό έχουν σημασία. Είτε βγαίνουν σωστά, είτε όχι."



Σ' αυτό είχε δίκιο. Θυμάμαι πολλές χρονιές τα αποτελέσματα να ξεκινούν με κάποιο λανθασμένο ή αποπροσανατολιστικό έξιτ πολλ, και μετά τα κόμματα να τρέχουν και να μη φτάνουν. Εχει πλάκα.

Στήθηκαν όλοι μπροστά στην τηλεόραση κι εγώ πήγα στην κουζίνα να πάρω ένα ποτήρι νερό.

"Εφτά με εννιά τοις εκατό;" είπε κάποιος για κάποιο από τα κόμματα. "Άλλο πάλι και τούτο."

"Πόσο παίρνει;" είπα εγώ ταραγμένος. "Τόσο ψηλά;"

"Ναι..." είπε διστακτικά η εξαδέλφη.

Το "ναι" αυτό ήταν για μένα η σταγόνα που έκανε το ποτήρι να ξεχειλίσει. Άφησα τους άλλους να βλέπουν τα έξιτ πολλ, πήρα τον φορητό της εξαδέλφης, πήγα στο μέσα δωμάτιο, συνδέθηκα στο ίντερνετ κι αγόρασα το πρώτο διαθέσιμο εισιτήριο προς Λονδίνο.

Ετσι βρέθηκα να είμαι εδώ σήμερα, στο Αναγνωστήριο Humanities I, στον πρώτο όροφο της Βρετανικής Βιβλιοθήκης.

Αγαπητοί μου,

Πριν από χρόνια είχα δει την ταινία "Ελέφαντας".

Ηταν του Gus van Sant. Επρόκειτο για μια σχεδόν ντοκυμενταρίστικη καταγραφή των γεγονότων κατά την διάρκεια μιας συγκεκριμένης ημέρας, σε ένα σχολείο κάπου στα βάθη της Αμερικής. Η μέρα εξελισσόταν με αργούς ρυθμούς και ενδεχομένως λίγο βαρετά -- όσο βαρετά φαντάζεται κανείς να εξελίσσεται μια μέρα σε ένα σχολείο κάπου στα βάθη της Αμερικής. Η κάμερα κοιτούσε λίγο από εδώ και λίγο από κει. Σιγά σιγά αναγνώριζες ένα δύο πρόσωπα, μυριζόσουν ένα ή δύο νήματα ιστορίας, καταλάβαινες κάποιες σχέσεις μεταξύ προσώπων, ένα φλερτ εδώ, μια αντιπάθεια εκεί. Διέκρινες κάποιες αναφορές σε προηγούμενα γεγονότα, ασαφείς όμως και αδιάφορες.

Στην πραγματικότητα, δεν συνέβαινε τίποτα άξιο λόγου.

Μέχρι που κάποια στιγμή δύο από τους χαρακτήρες έπαιρναν πιστόλια, πολυβόλα, και καραμπίνες, πήγαιναν στο σχολείο και, ξαφνικά, χωρίς καμιά προειδοποίηση, αρχινούσαν να μακελεύουν όσους συμμαθητές και δασκάλους έβλεπαν μπροστά τους.

Ετσι απλά. Μακελειό.

Στο τέλος η ταινία τελείωνε. Ετσι απλά. Χωρίς να εξηγήσει. Χωρίς να καταγγείλει. Χωρίς να προτείνει.

Ελέφαντας.

Ετσι νοιώθω σήμερα εδώ, καθισμένος στο Humanities I, γράφοντας με το μολύβι σε σελίδες Α4 που μου έχει δώσει ο ανιψιός.

Νοιώθω ότι στο σαλόνι των σπιτιών μας έχει μπει ένας Ελέφαντας.

Όσο ο Ελέφαντας παραμένει εκεί, και όσο το 9,40% του αυτού του λαού θα προτιμάει να τον αγνοεί, εγώ θα είμαι προσεκτικός.

Ναι. Εχω φοβηθεί.

Και τα βιβλία της ιστορίας στα ράφια του Humanities I δεν με καθησυχάζουν.

Σας ασπάζομαι

Θείος Ακάκιος

Skill: Intermediate
Tags: cracker, patcher, packer, import address table, anti-disassembly, anti-debugging, anti-dumping

PATCHER CRACKER PACKER

by Jef

Στη μνήμη του αγαπημένου φίλου
Αστέριου Παρλαμέντα

Προστασία από τους crackers!

Η φράση "ό,τι κλειδώνει ξεκλειδώνει" είναι γνωστή σε όλους. Η ερμηνεία που επιδέχεται όμως δεν είναι πάντοτε ίδια. Για κάποιους σημαίνει ότι το διαρκές κυνήγι της ασφάλειας είναι μάταιο και δεν έχει ενδιαφέρον. Αντίθετα, εμάς μας οπλίζει με πείσμα, για αδιάκοπη μελέτη και εξάσκηση. Σε αυτό το άρθρο θα πραγματοποιήσουμε μια μικρή εισαγωγή σε ζητήματα που αφορούν στην ασφάλεια των εφαρμογών. Ελπίζουμε να φέρει και σε σας την όρεξη για μελέτη.

Ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζουν οι εταιρείες παραγωγής λογισμικού είναι η πειρατεία. Κατά καιρούς έχουν εφεύρει διάφορους τρόπους για την καταπολέμησή της, αλλά τα αποτελέσματα δεν ήταν ποτέ τα επιθυμητά. Εξάλλου, πολλοί μεμονωμένοι προγραμματιστές υλοποιούν τις δικές τους μεθόδους προστασίας, που συνήθως σχετίζονται με την απαίτηση ενός σειριακού αριθμού. Τα μέτρα αυτού του είδους αποθαρρύνουν τους απλούς χρήστες που σκοπεύουν να χρησιμοποιήσουν παράνομα κάποιο πρόγραμμα, αλλά δεν προσφέρουν ουσιαστική προστασία. Όταν ασχοληθεί κάποιος cracker μ' ένα τέτοιο κλείδωμα, θα καταφέρει να το προσπεράσει πολύ εύκολα. Και κάπως έτσι, η κυκλοφορία μιας "σπασμένης" εκδοχής του προγράμματος είναι απλά θέμα χρόνου. Όπως αντιλαμβάνεστε, ο πόλεμος μεταξύ κατασκευαστών και crackers είναι αδιάκοπος: Κάθε φορά που οι κατασκευαστές εφευρίσκουν ή βελτιώνουν μια υπάρχουσα μέθοδο προστασίας, οι crackers εστιάζουν τα πυρά τους σε αυτήν και, αργά η γρήγορα, εμφανίζεται κάποιος που καταφέρνει να την ξεπεράσει.

Μέτρα προφύλαξης για τα μέτρα προφύλαξης

Αν και κανένα σύστημα προστασίας δεν είναι ανυπέρβλητο, οι κατασκευαστές δεν παίζουν με την ασφάλεια — ούτε ξεκινούν κάθε φορά από το μηδέν. Η συσσωρευμένη εμπειρία επιβάλλει την ενσωμάτωση δύο επιπέδων ασφάλειας: Αφενός υπάρχει ο κώδικας που ελέγχει αν ο χρήστης είναι νόμιμος, π.χ., ζητώντας κάποιον σειριακό αριθμό, τσεκάροντας για την παρουσία ειδικού hardware (HASP), ελέγχοντας αν χρησιμοποιείται το πρωτότυπο δισκάκι CD/DVD και πάει λέγοντας. Αφετέρου, υπάρχουν διάφορες τεχνικές που εμποδίζουν ή, τέλος πάντων, δυσκολεύουν την επέμβαση στον κώδικα του προγράμματος. Όπως αντιλαμβάνεστε, το δεύτερο επίπεδο ασφάλειας είναι εξαιρετικά κρίσιμο. Αν ο εκάστοτε cracker μπορεί να τροποποιήσει τον κώδικα ενός προγράμματος χωρίς δυσκολία, είναι σίγουρο ότι θα παρακάμψει όλους τους ελέγχους γνησιότητας αμέσως. Γι' αυτό το λόγο, οι κατασκευαστές φροντίζουν να προστατεύουν τα προϊόντα τους με τη βοήθεια ειδικών εργαλείων, που ονομάζονται patchers ή packers. Τα εν λόγω εργαλεία τροποποιούν τον κώδικα ενός προγράμματος με τέτοιο τρόπο, ώστε οποιαδήποτε επέμβαση στη λειτουργία του να είναι εξαιρετικά δύσκολη.

Αν και οι μεγάλοι κατασκευαστές αναπτύσσουν δικά τους εργαλεία, υπάρχουν αρκετές εταιρείες που ασχολούνται αποκλειστικά με τη δημιουργία patchers. Παρεμπιπτόντως, μία από τις καλύτερες εταιρίες του χώρου κατά το παρελθόν ήταν η Ελληνική "MLS Laserlock", με έδρα τη Θεσσαλονίκη. Για την κατασκευή τέτοιων εργαλείων απαιτείται άριστη γνώση της γλώσσας Assembly, καθώς και της δομής των εκτελέσιμων αρχείων που πρέπει να προστατευτούν. Επιπρόσθετα, οι δημιουργοί των patchers πρέπει να ξέρουν και τη δουλειά των crackers, ώστε να είναι σε θέση ν' αναπτύξουν αντίμετρα. Για να διευκολύνουμε τη μελέτη που ακολουθεί, νομίζουμε ότι θα είχε ενδιαφέρον να πούμε δυο λόγια για τον τρόπο εργασίας ενός cracker.

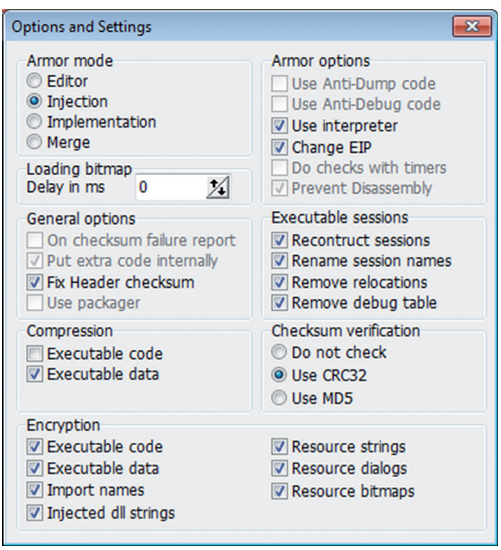
Ταχύρυθμο cracking!

Το σπάσιμο ενός *απλού* ελέγχου γνησιότητας είναι πανεύκολο. Τις περισσότερες φορές αρκεί η τροποποίηση ενός μόνο byte. Βλέπετε, κάθε έλεγχος που πραγματοποιείται σε ένα πρόγραμμα μεταφράζεται σε αυτό που ονομάζουμε conditional jump, δηλαδή σε μια εντολή σαν τη JZ (Jump if Zero) ή τη JNZ (Jump if Not Zero)[1].

1. Πρόκειται για εντολές που υπάρχουν σε κάθε παραλλαγή της Assembly. Μην ανησυχείτε, όμως: Δεν σκοπεύουμε να μπλέξουμε με τη συγκεκριμένη γλώσσα — τουλάχιστον όχι σ' αυτό το άρθρο ;)

Επομένως, για να παρακάμψουμε ένα τεστ γνησιότητας αρκεί να αντιστρέψουμε τη λογική του σχετικού ελέγχου. Με άλλα λόγια, αρκεί να μετατρέψουμε το JZ σε JNZ ή το αντίστροφο. Που βρίσκεται όμως αυτός ο έλεγχος;

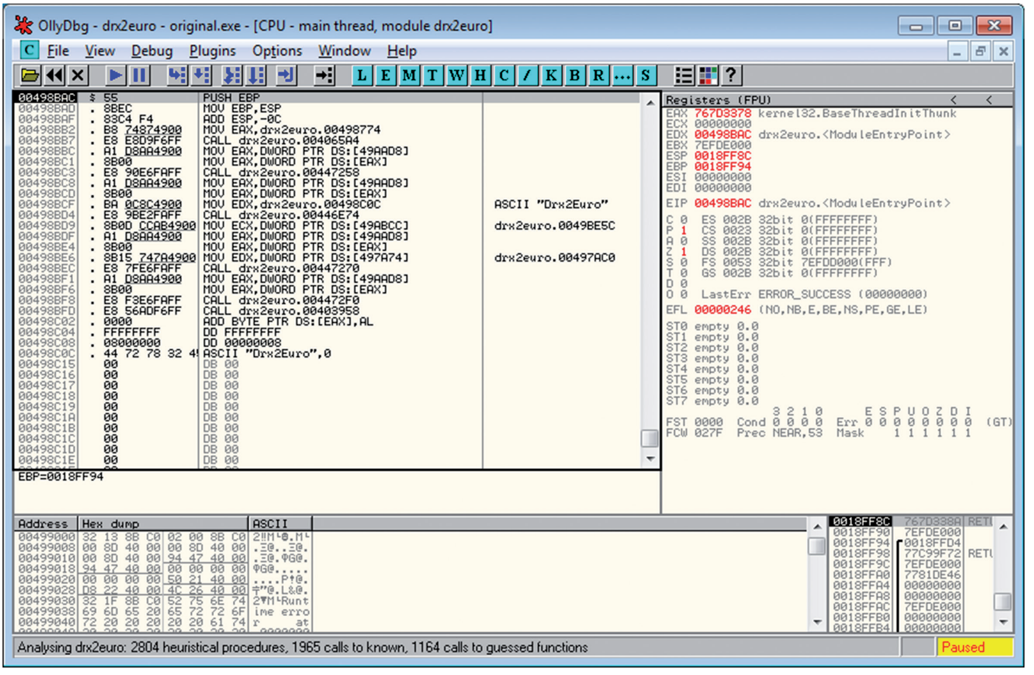
Συνήθως, ο εντοπισμός του ελέγχου που πρέπει να τροποποιήσουμε αποτελεί απλή υπόθεση. Αυτό συμβαίνει γιατί οι περισσότεροι προγραμματιστές ενημερώνουν το χρήστη για το αποτέλεσμα του ελέγχου, εμφανίζοντας το σχετικό μήνυμα σε ένα νέο παράθυρο. Στην περίπτωση των Windows, το εν λόγω παράθυρο είναι τύπου MessageBox και η εμφάνισή του πραγματοποιείται με μία κλήση στις αντίστοιχες βιβλιοθήκες του συστήματος. Για να πραγματοποιηθεί αυτή η κλήση, όμως, το πρόγραμμα ανατρέχει σ' έναν ειδικό πίνακα, όπου περιέχονται οι διευθύνσεις των βιβλιοθηκών και των συναρτήσεων. Αυτός ο πίνακας ονομάζεται Import Address Table (IAT) και υπάρχει σε κάθε εκτελέσιμο των Windows. Μη νομίζετε ότι αυτές οι λεπτομέρειες είναι άσχετες με το θέμα μας: Ακριβώς επειδή συμβαίνουν τα παραπάνω, ο cracker δεν έχει παρά να διακόψει την εκτέλεση του προστατευμένου προγράμματος, τη στιγμή που ο κώδικας ανατρέχει στον πίνακα IAT. Με αυτόν τον τρόπο το πρόγραμμα θα διακοπεί λίγο μετά την εκτέλεση του ελέγχου γνησιότητας κι ο cracker θα μπορεί να εντοπίσει τη σχετική εντολή πανεύκολα. Αν η αντιστροφή της εντολής έχει ως συνέπεια την ανεμπόδιστη λειτουργία του προγράμματος, το σπάσιμο θα είναι σχεδόν ολοκληρωμένο. Απομένει ο εντοπισμός της θέση της τροποποιημένης εντολής, μέσα στο εκτελέσιμο αρχείο. Αυτό είναι απαραίτητο για το μόνιμο σπάσιμο του προγράμματος, ώστε να μην απαιτείται κάθε φορά η επανάληψη της παραπάνω διαδικασίας.



Εικόνα 1
Οι ρυθμίσεις ενός patcher. Ο χρήστης μπορεί να καθορίσει πόσα συστήματα προστασίας θα ενεργοποιηθούν και κατ' επέκταση ποια θα είναι η επιβάρυνση κατά την εκτέλεση της προστατευμένης εφαρμογής.

Ένα πρόγραμμα, δύο αποστολές

Εξετάζοντας τη λειτουργία ενός patcher, θα μπορούσαμε να χωρίσουμε τον κώδικά του σε δύο κύρια τμήματα. Το ένα είναι αρκετά μεγαλύτερο από το άλλο κι αναλαμβάνει την ανάλυση και την τελική τροποποίηση του προγράμματος που θέλουμε να προστατεύσουμε. Το άλλο τμήμα είναι γραμμένο συνήθως σε Assembly και τοποθετείται αυτούσιο μέσα στο προστατευμένο πρόγραμμα. Το πρώτο κομμάτι οφείλει να εκτελέσει πολύ περισσότερες εργασίες. Καταρχάς πρέπει να εξετάσει το εκάστοτε πρόγραμμα, για να διαπιστώσει αν και σε ποια σημεία μπορεί να τροποποιήσει τον κώδικα. Η δομή ενός εκτελέσιμου εξαρτάται από τον compiler που το παρήγαγε. Τα εκτελέσιμα που παράγονται από το περιβάλλον .NET δεν μπορούν να δεχτούν αρκετές επεμβάσεις και η δουλειά των patchers γίνεται δύσκολη. Αντίθετα, τα προγράμματα που γράφτηκαν σε Delphi ή και σε C/C++, σηκώνουν αρκετές επεμβάσεις. Μετά την ανάλυση του

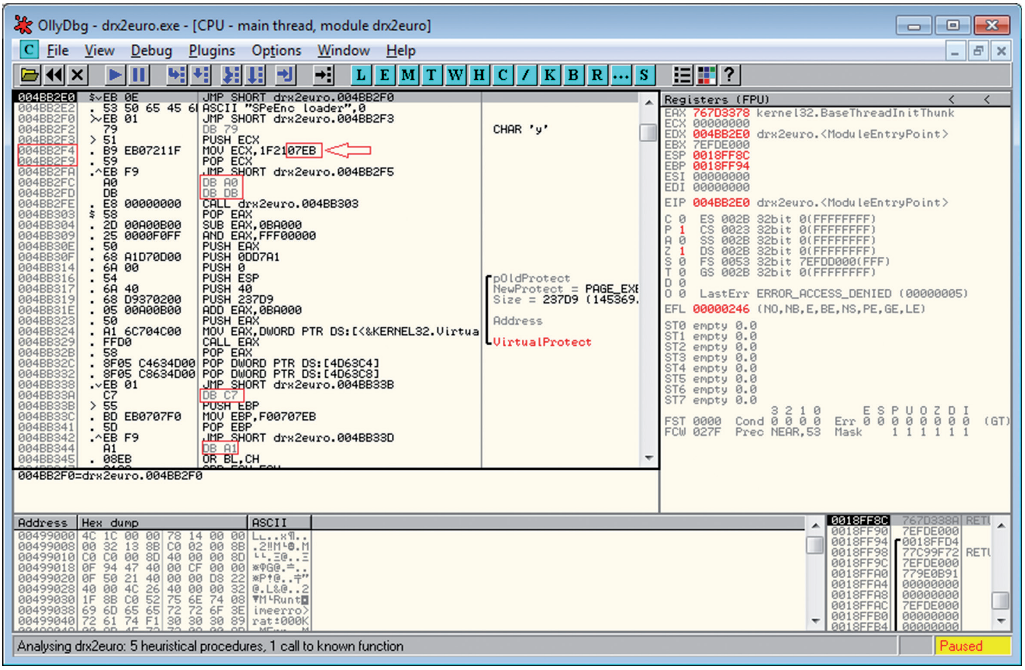


Εικόνα 2
Χωρίς την τεχνική anti-disassembly ο κώδικας του προγράμματος είναι ορατός στο σύνολό του και δεν μοιάζει να υπάρχει κάποιο πρόβλημα.

εκτελέσιμου ακολουθεί η τροποποίηση, με διάφορες τεχνικές που θα εξετάσουμε αργότερα. Στο τέλος των επεμβάσεων, το πρόγραμμα που θέλουμε να προστατεύσουμε είναι τόσο πολύ πειραγμένο και με τέτοιον τρόπο, που είναι αδύνατο να λειτουργήσει από μόνο του. Η εργασία του patcher ολοκληρώνεται με την εμφύτευση (injection) του δεύτερου τμήματος του κώδικά του, μέσα στο τροποποιημένο πρόγραμμα. Αυτό το τμήμα κώδικα λειτουργεί σαν διαμεσολαβητής ανάμεσα στο λειτουργικό σύστημα και στην υπό προστασία εφαρμογή που, όπως είπαμε, είναι αδύνατο πλέον να εκτελεστεί από μόνη της. Με άλλα λόγια, ο εμφυτευμένος κώδικας εξασφαλίζει την ικανότητα εκτέλεσης της τροποποιημένης εφαρμογής. Όπως υποψιάζεστε, αυτός ο τρόπος προστασίας επιδρά αρνητικά στην ταχύτητα εκτέλεσης μιας εφαρμογής, αφού ο κώδικάς της δεν εκτελείται απευθείας αλλά "διέρχεται" από τη μηχανή του patcher. Ακριβώς γι' αυτό, οι patchers επιτρέπουν στο χρήστη να επιλέξει εκείνος το πλήθος και το είδος τεχνικών που θα χρησιμοποιηθούν. Στην εικόνα 1 βλέπετε τις ρυθμίσεις ενός τυπικού patcher.

Τεχνικές προστασίας

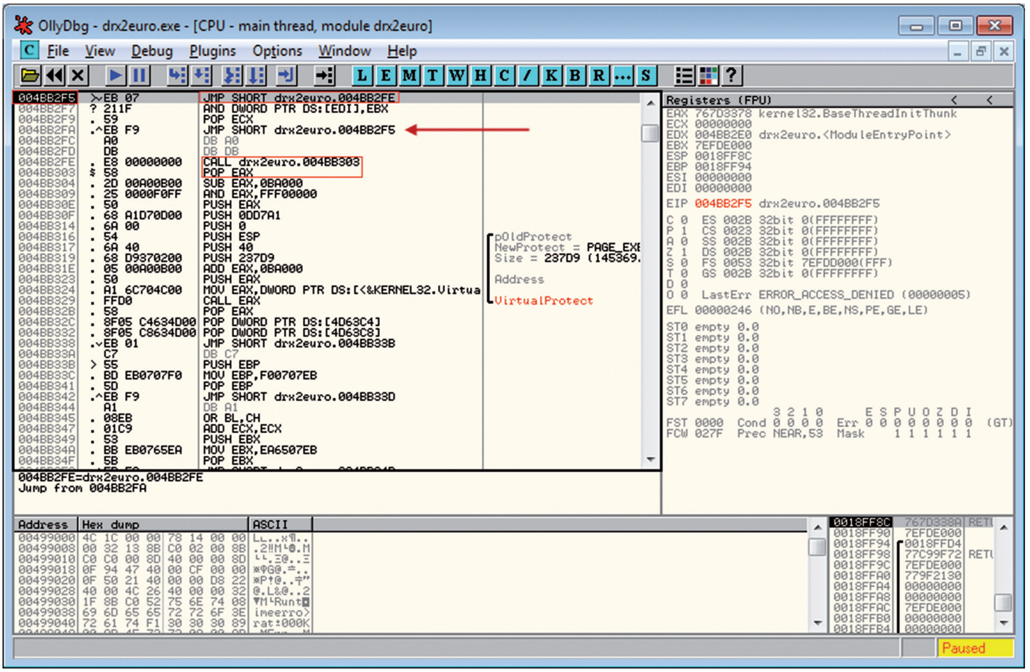
Οι τεχνικές που επιστρατεύουν οι patchers για την προστασία των εφαρμογών είναι γνωστές και λίγο-πολύ ίδιες. Αυτό που αλλάζει από patcher σε patcher είναι η υλοποίηση των τεχνικών, η "επιθετικότητα" που παρουσιάζουν κατά την εφαρμογή των τεχνικών κι ενδεχομένως το εύρος των επιλογών που προσφέρουν στο χρήστη. Στη συνέχεια θα ρίξουμε μια ματιά σε μερικές δημοφιλείς τεχνικές.



Εικόνα 3

Το anti-disassembly έχει αρχίσει να δημιουργεί προβλήματα. Καταρχάς, σε πολλά σημεία του κώδικα ο debugger απέτυχε να αποκωδικοποιήσει κάποια εντολή κι εμφανίζει κάποιους ασυνάρτητους χαρακτήρες. Παρατηρήστε επίσης ότι στη διεύθυνση 4BB2F4 υπάρχει η εντολή "MOV ECX, 1F2107EB" — ή τέλος πάντων έτσι νομίζει ο debugger. Πάντως στη διεύθυνση 4BB2F5 δεν υπάρχει κάποια εντολή... :/

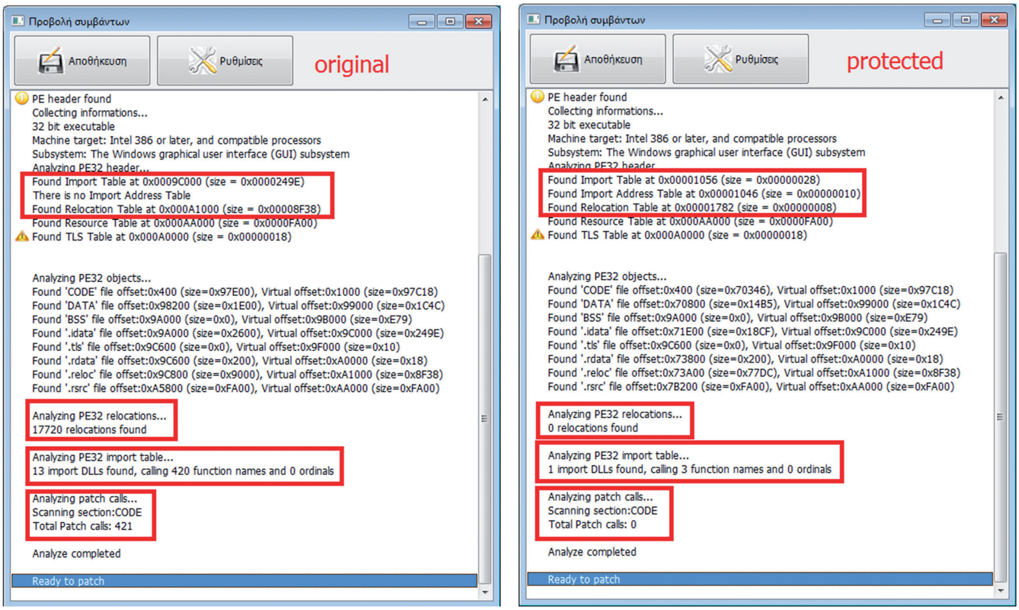
- **Anti-disassembly:** Αυτή η τεχνική δεν συνιστά ιδιαίτερα δύσκολο εμπόδιο, αλλά είναι σίγουρο ότι προκαλεί μεγάλη καθυστέρηση στη δουλειά του cracker. Το anti-disassembly επιτυγχάνεται με την εισαγωγή μιας μικρής αλληλουχίας χαρακτήρων σε διάφορα σημεία του κώδικα. Οι χαρακτήρες που προστίθενται "αποπροσανατολίζουν" τον debugger κι ως αποτέλεσμα το disassembly ολόκληρου του προγράμματος είναι αδύνατο. Ο εμφυτευμένος κώδικας του patcher απομακρύνει τους περιττούς χαρακτήρες μόνο από την περιοχή που πρόκειται κάθε φορά να εκτελεστεί και στη συνέχεια τους προσθέτει ξανά. Έτσι, ο cracker μπορεί κάθε φορά να βλέπει ένα μικρό τμήμα του κώδικα κι όχι το σύνολό του. Αυτό δυσκολεύει την κατανόηση της ροής του κώδικα, αλλά δεν την καθιστά κι αδύνατη. Με αρκετό κόπο και χρόνο, ο cracker μπορεί να χρησιμοποιήσει τα επιμέρους τμήματα για να συνθέσει ολόκληρο τον κώδικα και να καταλάβει τι γίνεται και πού. Για να πάρετε μια γεύση όσων καταφέρνει το anti-disassembly, δείτε τις εικόνες 2, 3 και 4.
- **Anti-debugging.** Πρωταρχικός σκοπός αυτής της τεχνικής είναι και πάλι η πρόκληση σύγχυσης, ώστε η δουλειά του cracker να καθυστερήσει όσο περισσότερο γίνεται. Επί της ουσίας, επιδιώκουμε να δυσκολέψουμε την παρακολούθηση της ροής του κώδικα. Για να πετύχουμε κάτι τέτοιο, ένα κρίσιμο τμήμα κώδικα που υπό άλλες συνθήκες θα τοποθετούσαμε σε μία απλή συνάρτηση, το εισάγουμε στη ρουτίνα εξυπηρέτησης ενός σφάλματος (exception handler). Έτσι, οποτεδήποτε θέλουμε να εκτελέσουμε το συγκεκριμένο τμήμα κώδικα, δεν πραγματοποιούμε μια τυπική κλήση με την εντολή call αλλά προκαλούμε



Εικόνα 4

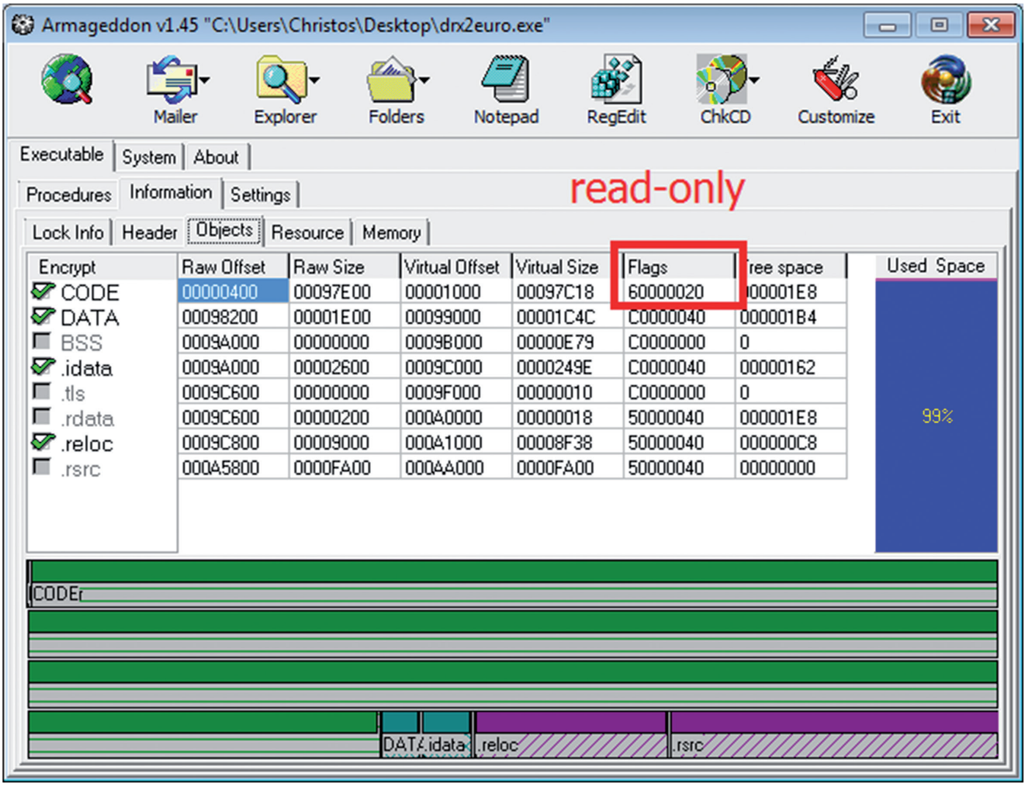
Τελικά υπάρχουν περισσότερα προβλήματα απ' όσα φάνηκαν αρχικά. Στη διεύθυνση 4BB2FA πραγματοποιείται ένα JMP προς τη διεύθυνση 4BB2F5. Εκεί βλέπουμε ότι υπάρχει η εντολή "JMP $+7", με opcode το "EB 07". Στην προηγούμενη εικόνα, όμως, βλέπαμε ότι στη συγκεκριμένη διεύθυνση δεν υπάρχει εντολή. Μάλιστα το opcode της εντολής (EB 07) εμφανιζόταν σαν τμήμα της εντολής της προηγούμενης διεύθυνσης. Όπως καταλαβαίνετε, ο debugger έχει χάσει τ' αυγά και τα πασχάλια!

κάποιο σφάλμα (exception) στην εκτέλεση του προγράμματος. Με αυτόν τον τρόπο είναι δύσκολο να προβλέψει κανείς τη ροή του κώδικα. Ως παράδειγμα, σκεφτείτε την περίπτωση που θέλουμε το πρόγραμμά μας να τερματίζεται όταν εκτελείται μέσα σε debugger ή σε κάποια εικονική μηχανή. Αν χρησιμοποιούσαμε έναν απλό έλεγχο και μια κλήση της ExitProcess, η σχετική λειτουργία θα γινόταν αμέσως αντιληπτή από έναν cracker. Αντί για κάτι τόσο φανερό, λοιπόν, μπορούμε να δημιουργήσουμε έναν exception handler ο οποίος θα καλεί την ExitProcess. Πλέον, όταν η εφαρμογή μας εκτελείται υπό παρακολούθηση ο κώδικας δεν θα πραγματοποιεί κάποια κλήση, αλλά θα προκαλεί ένα exception. Με αυτόν τον τρόπο, ο τερματισμός του προγράμματος θα γίνεται ύπουλα. Η πρόκληση ενός exception είναι ιδιαίτερα απλή υπόθεση. Ένα από τα πιο συνηθισμένα τρικ αποτελεί η χρήση μιας συγκεκριμένης διακοπής (interrupt) με την εντολή "INT 3" (0xCC). Πρόκειται για το interrupt που χρησιμοποιούν συνήθως οι debuggers, για τη διακοπή της εκτέλεσης ενός προγράμματος. Ένας άλλος τρόπος είναι η πρόκληση του σφάλματος "access violation", που μπορεί να επιτευχθεί με αρκετούς και εύκολους τρόπους, όπως, π.χ., με τη διαδοχική εκτέλεση των "mov ebx, 0" και "mov [ebx], ebx". Όπως αντιλαμβάνεστε, η παρακολούθηση των exceptions είναι αρκετά δύσκολη και προκαλεί σημαντική καθυστέρηση στη δουλειά ενός cracker. Μάλιστα, αν το πρόγραμμά μας είναι multi-threaded και παράγονται διάφορα exceptions από διαφορετικά threads, ο επίδοξος creacker είναι πολύ πιθανό να χάσει το μπούσουλα.



Εικόνα 5
Όπως βλέπετε, το import table αναφερόταν αρχικά σε 13 DLL και 420 συναρτήσεις. Στην προστατευμένη εκδοχή του προγράμματος, ο ίδιος πίνακας διαθέτει μόλις μία DLL με 3 συναρτήσεις. Φυσικά, οι υπόλοιπες συναρτήσεις εξακολουθούν να υφίστανται και είναι κρυμμένες σε περιοχές που γνωρίζει ο εμφυτευμένος κώδικας του patcher (στη συγκεκριμένη περίπτωση εμφανίζονται "overlay").

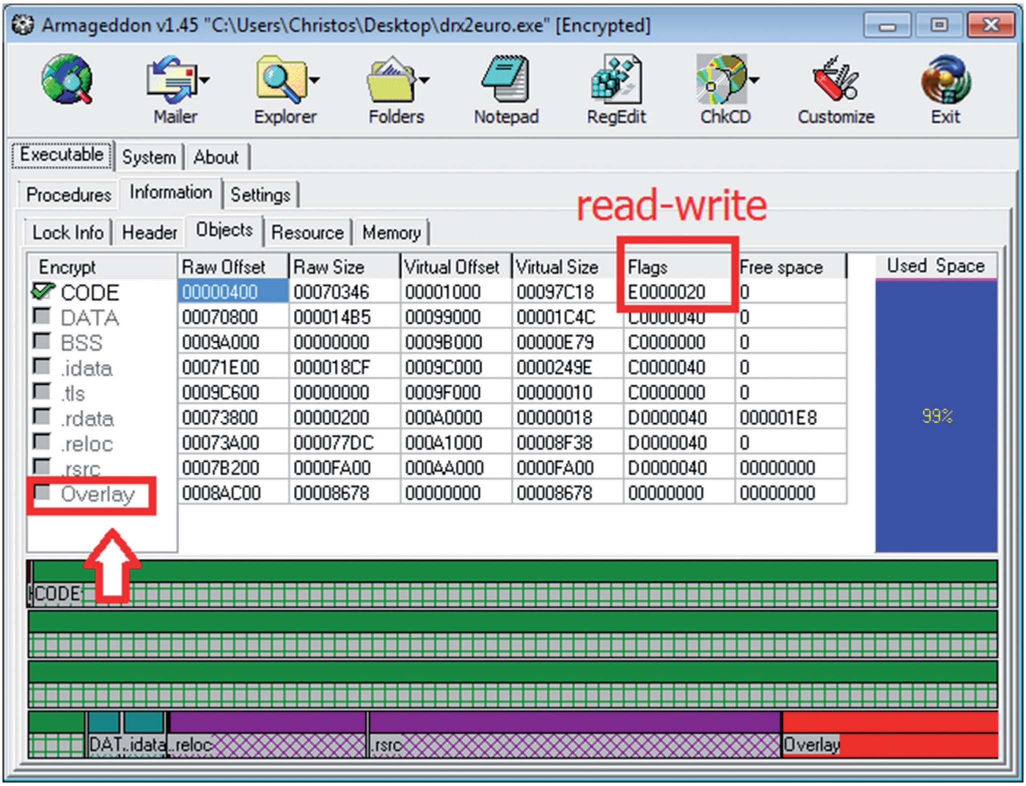
- **Anti-dumping.** Το ζητούμενο με αυτή την τεχνική είναι να αποτρέψουμε τη μεταφορά του κώδικα του προγράμματος από τη μνήμη στο δίσκο. Υπάρχουν αρκετά αξιόλογα εργαλεία που πραγματοποιούν το dumping με επιτυχία. Μάλιστα, ορισμένα από αυτά διορθώνουν και τις πιθανές αλλοιώσεις στον πίνακα IAT. Επομένως, ο σχετικός κίνδυνος είναι υπαρκτός και μεγάλος. Για την αποτροπή του dumping υπάρχουν διάφορα τρικ, τα οποία παρέχουν προστασία κι από άλλες επιθέσεις. Το μυστικό έγκειται στην τοποθέτηση των κρίσιμων τμημάτων του κώδικα σε διαφορετικές θέσεις μνήμης, κάθε φορά που εκτελείται η εφαρμογή μας. Προφανώς, οι θέσεις αυτές δεν πρέπει να είναι προκαθορισμένες και γνωστές! Αυτό σημαίνει ότι τα κρίσιμα τμήματα της εφαρμογής μας είναι κατάλληλα γραμμένα, ώστε να εκτελούνται σωστά από οποιαδήποτε περιοχή της μνήμης. Επιπρόσθετα, τα ίδια τμήματα του κώδικα πρέπει να είναι κρυπτογραφημένα και να μην εμφανίζονται στην κανονική τους μορφή, πουθενά μέσα στο εκτελέσιμο αρχείο. Τέλος, η εφαρμογή μας πρέπει να περιλαμβάνει και ένα τμήμα κώδικα που θα δεσμεύει χώρο στη μνήμη (σε τυχαία θέση), θα αποκρυπτογραφεί τα κρίσιμα τμήματα του κώδικα και θα τα τοποθετεί στη μνήμη!
- **Τροποποίηση του IAT.** Όπως είπαμε νωρίτερα, τα προγράμματα διατηρούν έναν πίνακα με τις διευθύνσεις των απαιτούμενων βιβλιοθηκών και διαφόρων συναρτήσεων. Πρόκειται για τον πίνακα "Import Address Table", χωρίς τον οποίο είναι αδύνατο να λειτουργήσει κάποιο πρόγραμμα. Όπως αντιλαμβάνεστε, παραποιώντας τα περιεχόμενα αυτού του πίνακα ή και κρύβοντάς τα, η δουλειά του cracker θα γίνει από δύσκολη έως αδύνατη. Ένα τέχνασμα



Εικόνα 6
Τα περισσότερα τμήματα σε ένα κανονικό εκτελέσιμο επιτρέπουν μόνο την ανάγνωση, ιδιαίτερα το τμήμα που περιέχει τον κώδικα (το συναντάμε ως .text ή CODE). Έστω κι ένα byte να αλλάξει εκεί, προκαλείται access violation…

που επιστρατεύεται εδώ, είναι παρόμοιο με αυτό που συναντήσαμε στο anti-dumping: Σε κάθε εκτέλεση του προγράμματος ο πίνακας IAT τοποθετείται σε μία νέα θέση. Στην αρχική/προβλεπόμενη θέση του πίνακα μπορούμε να τοποθετήσουμε μερικές τυχαίες διευθύνσεις μαζί με δυο-τρεις σωστές, ώστε ο παροπλισμένος πίνακας να μοιάζει έγκυρος. Επιπρόσθετα, μέσα στον αληθινό πίνακα μπορούμε να τοποθετήσουμε και μερικές παραπομπές προς τον εμφυτευμένο κώδικα του patcher. Στις σχετικές θέσεις του εμφυτευμένου κώδικα θα βρίσκονται ρουτίνες που πραγματοποιούν ανακατεύθυνση των κλήσεων, ώστε η εκτέλεση του προγράμματος να συνεχίζεται κανονικά, ενώ ο cracker να μπερδεύεται όλο και περισσότερο ;)

- **Κρυπτογράφηση.** Για το τέλος αφήσαμε την κρυπτογράφηση, που αν και δεν αποτελεί πανάκεια είναι ιδιαίτερα σημαντική. Βλέπετε, η κρυπτογράφηση αξιοποιείται σε όλες σχεδόν τις τεχνικές που αναφέραμε προηγουμένως. Εδώ υπάρχει μια κρίσιμη λεπτομέρεια: Το πρόγραμμά μας θέλουμε να εκτελείται χωρίς να ζητάει από το χρήστη κάποιο κλειδί αποκρυπτογράφησης — έτσι δεν είναι; Τώρα όμως μπορεί να σκέφτεστε ότι δεν έχει κανένα νόημα να κρυπτογραφήσουμε κάτι, εφόσον θα δώσουμε μαζί και το κλειδί αποκρυπτογράφη-



Εικόνα 7
Σε ένα προστατευμένο εκτελέσιμο συμβαίνει το αντίθετο: Τα περισσότερα τμήματά του είναι εγγράψιμα. Αυτό συμβαίνει ώστε κατά την εκτέλεση του προγράμματος να είναι δυνατή η αποκρυπτογράφηση και η εγγραφή πάνω σε περιοχές του ίδιου του κώδικα!

σης. Τα πράγματα όμως δεν είναι έτσι. Σκεφτείτε την περίπτωση που έχουμε χρησιμοποιήσει ως κλειδί αποκρυπτογράφησης το CRC checksum του εμφυτευμένου κώδικα του patcher. Αν ο cracker αλλάξει έστω και ένα byte από τον κώδικα του patcher, το κλειδί αποκρυπτογράφησης θα είναι άκυρο.

Όπως καταλαβαίνετε από την όλη παρουσίαση, μία μέθοδος προστασίας δεν είναι ποτέ αρκετή. Για να μπορούμε να πούμε ότι ένας patcher είναι πραγματικά αξιόλογος, χρειάζεται να υλοποιεί όλα τα τεχνάσματα που αναφέραμε καθώς και πάρα πολλά ακόμα. Προκειμένου να αντιληφθούμε το πλήθος των εργασιών που πραγματοποιεί ένας patcher, μπορούμε να συγκρίνουμε την αρχική μορφή ενός προγράμματος με την προστατευμένη (εκείνη που παράγει ο patcher). Στις περισσότερες περιπτώσεις, τα δύο εκτελέσιμα θα μοιάζουν να αποτελούν εντελώς διαφορετικά προγράμματα! Πριν κλείσουμε προσωρινά το θέμα των patchers —και λέμε "προσωρινά" διότι σκοπεύουμε να επανέλθουμε με αναλυτικότερα άρθρα και κώδικα— αξίζει να ρίξουμε μια ματιά και στο αρχείο καταγραφής ενός patcher. Σας αφήνουμε να χαζέψετε τη σχετική εικόνα, πιστεύοντας ότι καταφέραμε να κερδίσουμε το ενδιαφέρον σας για τα καλά. Ο τομέας της προστασίας των προγραμμάτων παρουσιάζει πολλές προκλήσεις και προϋποθέτει ακόμα περισσότερες γνώσεις. Βέβαια το τέλειο σύστημα προστασίας δεν θα υπάρξει ποτέ. Αυτό σημαίνει ότι η μελέτη του συγκεκριμένου τομέα δεν τερματίζεται ποτέ, ενώ θα προσφέρει πάντοτε και νέες γνώσεις.

patcher.log - Notepad

File Edit Format View Help

```
Coping import table...
Analyzing Import Table
Total bytes needed for import table: 9214 for 13 DLLs
Saving Import Table informations
Scaning DLLs...
Renaming 9 sections...
Renaming Section name from '.ldata ' to 'Object1'
Renaming Section name from 'DATA' to 'Object2'
Renaming Section name from 'BSS' to 'Object3'
Renaming Section name from '.idata' to 'Object4'
Renaming Section name from '.tls' to 'Object5'
Renaming Section name from '.rdata' to 'Object6'
Renaming Section name from '.reloc' to 'Object7'
Renaming Section name from '.rsrc' to 'Object8'
Renaming Section name from '.lcode' to 'Object9'
Done
Merging buffers...
Coping main buffer...
Coping patcher buffer...
Coping import table (9214 bytes)...
Creating Import Table at 0x000DB000...
Relocating patch calls from 0x0009C000 to 0x000DB000
Scanning section 'Object1'...
421 patch calls relocated
Cleaning old Import Table
Encrypting original Import Table
Changing Import table from 0x0009C000 to 0x000C8C14 (size=0x00000050)
Changing Import address table from 0x00000000 to 0x000C8C14 (size=0x00000050)
Changing import table entries from 0x0000EC14 to 0x000C8C14...
Done
Coping encrypted strings array...
Coping encrypted resources array...
Compressing data...
Object2 compressed from 7244 bytes to 5240 bytes
Object4 compressed from 9374 bytes to 2593 bytes
Object6 ignored
Object7 ignored
Object8 ignored
Encrypting old executable code (Offset:00000400, Size:00097C18)
Coping parameters struct (size=56)
Scanning injected DLL for 'ReturnToOldEIP' signature...
Found signature at 0x00001ED5
Scanning injected DLL for 'StartUP' signature...
Found signature at 0x000012FE
Changing EIP from 0x00098BAC to 0x000BB2E0
Setting Fixup RVA 0x000BA000
Setting parameters RVA 0x000DD7A1
Setting Injection object size 0x000237D9
Setting Injection RVA 0x000BA000
Setting JmpToOldEIP
Fixing Nops...
161 Total Nops patched
Scanning 4nd encryption level signature...
Encrypting 124 bytes...
Scanning 3nd encryption level signature...
Encrypting 525 bytes...
Scanning 2nd encryption level signature...
Encrypting 10235 bytes...
Scanning 1st encryption level signature...
Encrypting 78 bytes...
Done
```

Εικόνα 8

Ο patcher που δημιούργησε το συγκεκριμένο αρχείο καταγραφής μετονόμασε τα section names, τους άλλαξε μέγεθος και θέση, κρυπτογράφησε τον κώδικα σε πολλαπλά επίπεδα, κατήργησε τον αληθινό πίνακα relocation και τον αντικατέστησε από έναν ψεύτικο, ενώ στο τέλος ενσωμάτωσε και μια DLL, που αποτελεί τον εμφυτεύσιμο κώδικα... Παιχνιδάκι ;)

Skill: Intermediate
Tags: port knocking, iptables



by Spir@lEvolut10n

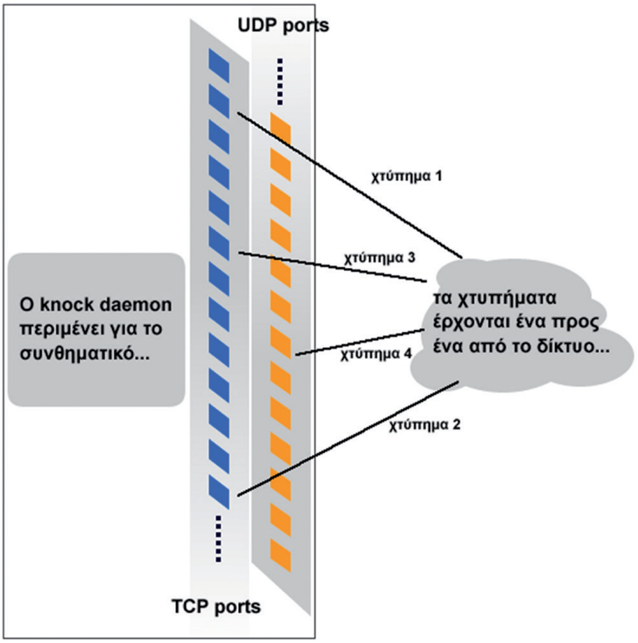
Port knocking part 1

Στήστε αφτί ...στη δική σας πόρτα

Όσες αμπάρες και να βάλουμε, όσα λουκέτα κι όσους σύρτες, η πόρτα μας δεν θα είναι ποτέ ασφαλής. Όσο δίνουμε στόχο, τόσο θα προσπαθούν να μπουν και καθόλου δεν αποκλείεται κάποια στιγμή να τα καταφέρουν! Μήπως λοιπόν να κρύβαμε την πόρτα καλύτερα;

Δεν έχει καμία σημασία τι κάνει το μηχάνημα ή το VPS μας στο δίκτυο. Τις περισσότερες φορές, μάλιστα, δεν έχει καμία σημασία το ποιοι είμαστε και το αν μας γνωρίζουν. Σημασία έχει το ότι είμαστε δικτυωμένοι. Αυτό και μόνο αποτελεί καλό λόγο για κάθε "περίεργο" (εντός κι εκτός εισαγωγικών) να μας επιτεθεί με όποιες μεθόδους γνωρίζει — ή με όποιο script ξετρύπωσε και θέλει να δοκιμάσει, τέλος πάντων. Έτσι, είτε διαθέτουμε έναν πολυάσχολο server, είτε ένα μικρό VPS που προσφέρει μια, δυο υπηρεσίες, θα ήταν φρόνιμο να φυλαγόμαστε. Φυσικά, σαν προσεκτικοί administrators που (θέλουμε να) είμαστε, έχουμε ρυθμίσει προσεκτικά όλες τις υπηρεσίες και φυσικά το firewall. Ας μη γελιόμαστε, όμως. Η σωστή ρύθμιση κάθε παραμέτρου ούτως ή άλλως αποτελεί αυτονόητο καθήκον ακόμη και για έναν μέσο χρήστη. Μήπως λοιπόν πρέπει να κάνουμε κάτι παραπάνω; Όπως πάντα, όσοι χρησιμοποιούμε Linux έχουμε πολλές επιλογές. (Σ.τ.Ε. Επ! Κι όλοι όσοι χρησιμοποιούμε Unix-οειδές OS, γενικότερα :D)

Ένας δραστήριος administrator μπορεί να σκαρφιστεί χίλια δυο μέτρα για να προφυλάξει το μηχάνημά του. Μία καλή ιδέα, όταν γνωρίζουμε ότι κάποιες υπηρεσίες χρησιμοποιούνται μόνο σε συγκεκριμένες ώρες, είναι να τις απενεργοποιούμε για όλο το υπόλοιπο διάστημα της ημέρας. Με αυτόν τον τρόπο, τα πιθανά κενά στην ασφάλεια του συστήματος θα εκτίθενται στο δίκτυο για τον ελάχιστο δυνατό χρόνο. Μικραίνουμε, μ' άλλα λόγια, τη λεγόμενη attack surface. Εντάξει, αν και αυτό το μέτρο είναι ολίγον τι ...ημίμετρο, είναι σίγουρα αρκετά αποτελεσματικό. Ωστόσο, τις ώρες που οι υπηρεσίες μας ενεργοποιούνται, οι κίνδυνοι επανέρχονται. Υπάρχουν βέβαια κι άλλα τερτίπια, πολλά από τα οποία είναι εξαιρετικά πιο ευρηματικά. Ένα από αυτά ακούει στο όνομα port knocking: Με τη βοήθειά του, οι υπηρεσίες που προσφέρει το μηχάνημά μας μπορούν να κρύβονται συνεχώς και να αγνοούν τα αιτήματα των αγνώστων, ακόμα κι όταν εκτελούνται κανονικά. (Σ.τ.Ε. Οι υπηρεσίες, όχι οι άγνωστοι :P)



Η ιδέα του τεχνάσματος port knocking είναι γνώριμη από την πραγματική ζωή: Το προγραμματάκι που τρέχει στο σύστημά μας, περιμένει να δεχτεί τη σωστή ακολουθία "χτυπημάτων" στα δικτυακά ports. Όταν και αν συμβεί αυτό, εκτελεί μία προκαθορισμένη εντολή…

Δεν είμαι εδώ

Κάθε ενεργοποιημένη υπηρεσία ακούει από ένα συγκεκριμένο port ή, γενικότερα, από ένα συγκεκριμένο σύνολο ports. Άλλωστε, ακριβώς αυτά ανοίγουμε ή κλείνουμε με το firewall, προκειμένου να επιτρέπουμε ή να απαγορεύουμε την πρόσβαση σε υπηρεσίες κι εφαρμογές "πίσω" τους. (Παρεμπιπτόντως, για το τι ακριβώς είναι τα ports μπορείτε να διαβάσετε στο σχετικό άρθρο που ξεκινά από τη σελίδα 24, του deltaHacker 018.) Τώρα, για να μην ξεφεύγουμε από το θέμα μας, να πούμε ότι το port knocking προβλέπει κατ' αρχάς το κλείσιμο όλων των ports των υπηρεσιών που θέλουμε να προφυλάξουμε. Στη συνέχεια, το πρόγραμμα που υλοποιεί το port knocking παρακολουθεί τα (κλειστά, πλέον) ports και περιμένει να δεχτεί ένα συνθηματικό "χτύπημα" που εμείς έχουμε προκαθορίσει. Όταν συμβεί κάτι τέτοιο, πραγματοποιεί μία ενέργεια που επίσης έχουμε υπαγορεύσει και η οποία, συνήθως, επιβάλλει το άνοιγμα κάποιου port. Έτσι, ενώ αρχικά τα ports του υπολογιστή είναι κλειστά, αρκεί να "χτυπήσουμε" μ' έναν συγκεκριμένο τρόπο. Κάνοντας κάτι τέτοιο, κάποιο από τα ports θα ανοίξει και θα αποκαλύψει μία κρυμμένη υπηρεσία. Για παράδειγμα, φανταστείτε ότι πάτε στο σπίτι ενός ιδιότροπου φίλου, ο οποίος δεν θέλει να δει κανέναν εκτός από εσάς. Αν θέλετε να σας αναγνωρίσει και ν' ανοίξει (χωρίς να πάρετε τηλέφωνο, να στείλετε μήνυμα κ.λπ.), θα πρέπει να χτυπήσετε το κουδούνι ή την πόρτα με έναν προσυμφωνημένο τρόπο. Ακριβώς αυτό το κολπάκι υλοποιεί το port knocking. Μόνο που στην περίπτωσή του, τα χτυπήματα στην πόρτα αντικαθίστανται από δικτυακές αιτήσεις για σύνδεση (εισερχόμενα δικτυακά πακέτα σε συγκεκριμένα ports), που πέφτουν πάνω σε ένα κλειστό firewall. Στη συνέχεια θα δούμε τι πρέπει να κάνουμε, προκειμένου να προστατεύσουμε και το δικό μας μηχάνημα/VPS με μία υλοποίηση του port knocking.

```
pvar@crunchbang: ~
  GNU nano 2.2.6              File: /etc/knockd.conf

[options]
        UseSyslog

[openSSH]
        sequence    = 7000,8000,9000
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn

[closeSSH]
        sequence    = 9000,8000,7000
        seq_timeout = 5
        command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn

                              [ Read 15 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Το αρχείο ρυθμίσεων του knockd, με τις "εργοστασιακές" ρυθμίσεις. Η δεύτερη και η τρίτη ενότητα ορίζουν δύο συνθηματικά. Όταν λαμβάνεται το πρώτο συνθηματικό, το knockd ανοίγει το port 22 στο firewall, ενώ στην άλλη περίπτωση ξανακλείνει το port 22. Την πρώτη ενότητα μπορείτε να την αγνοήσετε με ασφάλεια, αφού καθορίζει την υπηρεσία για διατήρηση των log files και τίποτα παραπάνω.

Τα προκαταρκτικά

Προφανώς, πριν κάνουμε οτιδήποτε άλλο θα πρέπει να εγκαταστήσουμε τα απαιτούμενα εργαλεία. Το ένα από αυτά είναι εκείνο που περιμένει τα συνθηματικά κι αποκρίνεται κατάλληλα (daemon), ενώ το άλλο είναι εκείνο με το οποίο πραγματοποιούμε τα χτυπήματα (client). Όσοι χρησιμοποιούν Debian-based διανομή, με δικαιώματα διαχειριστή, αρκεί να πληκτρολογήσουν σε ένα τερματικό

```
apt-get update
```

και μετά

```
apt-get install knockd
```

Σε αυτό το σημείο, πρέπει να υπογραμμίσουμε μια λεπτομέρεια. Το πακέτο που αναφέραμε (knockd) θα πρέπει να εγκατασταθεί στο μηχάνημα που θέλουμε να προφυλάξουμε, αλλά και στα μηχανήματα από τα οποία θα πραγματοποιούμε τις συνδέσεις. Για την εγκατάσταση σε συστήματα με Windows, πρέπει να κατεβάσουμε το σχετικό πακέτο (Native Win32 Client) από τη διεύθυνση www.zeroflux.org/projects/knock. Ωστόσο, σημειώστε ότι για τα Windows προσφέρεται μόνο το πρόγραμμα που πραγματοποιεί τα χτυπήματα (client).

Αφού ολοκληρώσουμε την εγκατάσταση συνεχίζουμε με τη ρύθμιση του firewall, το οποίο είναι το iptables. Όπως είπαμε παραπάνω, η κεντρική ιδέα είναι να κλείσουμε όλα τα ports των υπηρεσιών που θέλουμε να προφυλάξουμε. Το συντακτικό του iptables ξεφεύγει από τους σκοπούς του άρθρου και δεν θα επεκταθούμε ιδιαίτερα. (Σ.τ.Ε. Διαβάστε το σχετικό άρθρο που ξεκινά από τη σελίδα 36 του deltaHacker 014.) Ακριβώς γι' αυτό, θα εξετάσουμε την εξαιρετικά απλουστευμένη περίπτωση κατά την οποία θέλουμε να κλείσουμε *όλα* τα ports. Σ' αυτό το σημείο, όμως, πρέπει να επιστήσουμε την προσοχή σας σε κάτι σημαντικό: Αν δουλεύετε σε κάποιο μηχάνημα απομακρυσμένα και κλείσετε όλα τα ports, θα κλειδωθείτε απέξω. Μπορεί βέβαια να σας παρέχεται κάποια web-based VNC console, όμως αυτό δεν είναι κι ο κανόνας. Για το λόγο αυτό, θα ήταν φρονιμότερο να κάνετε τις δοκιμές σας *πρώτα* σε κάποιο virtual machine ;) Ας δούμε τώρα το κλείσιμο όλων των θυρών:

```
iptables -F
iptables –X
```

Δίνοντας αυτές τις παραμέτρους στο iptables (-F και -X), θα διαγραφούν οι πίνακες του firewall και όλοι οι κανόνες θα καταργηθούν. Συνεχίζουμε κάπως έτσι:

```
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
```

Μετά απ' αυτά, η προκαθορισμένη συμπεριφορά για το iptables θα είναι να αφήνει όλα τα εξερχόμενα πακέτα και να απορρίπτει όλα τα εισερχόμενα. Αυτή η ρύθμιση όμως είναι μάλλον βάρβαρη, ακόμα και για τα δικά μας μέτρα. Επομένως, καλό θα ήταν να δώσουμε και τα ακόλουθα:

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```

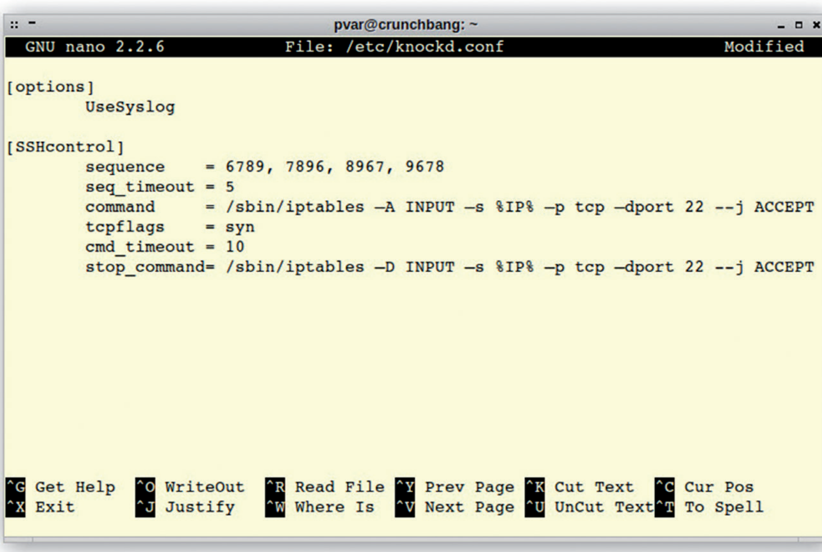
Στην πρώτη γραμμή ορίζουμε έναν κανόνα που δηλώνει ότι ορισμένα εισερχόμενα πακέτα θα πρέπει να διέρχονται ανεμπόδιστα. Για την ακρίβεια, διέρχονται τα πα-

κέτα που ανήκουν ή σχετίζονται με κάποια σύνδεση που πραγματοποιήσαμε εμείς[1]. Εκτελώντας τη δεύτερη γραμμή, επιβάλλουμε στο iptables να δέχεται αδιακρίτως τα εισερχόμενα πακέτα που προέρχονται από το ίδιο το σύστημα. Αυτός ο κανόνας προστίθεται για την ομαλή λειτουργία ορισμένων δικτυακών εφαρμογών, που ακόμα κι όταν χρησιμοποιούνται τοπικά επικοινωνούν με τις άλλες εφαρμογές χρησιμοποιώντας δικτυακά πρωτόκολλα.

Μυστικά σινιάλα

Μετά από όσα είδαμε, τα απαραίτητα εργαλεία είναι εγκατεστημένα και το firewall έχει κλείσει όλα τα ports. Πλέον, μπορούμε να προχωρήσουμε στο ζουμί της υπόθεσης: Στη ρύθμιση του knock daemon. Γι' αυτή την εργασία θα επέμβουμε σε ένα αρχείο κειμένου, που περιλαμβάνει όλες τις ρυθμίσεις για το knockd. Εκεί, με τη βοήθεια ενός απλού συντακτικού, θα περιγράψουμε τους επιθυμητούς συνδυασμούς χτυπημάτων καθώς και τις αντίστοιχες ενέργειες που θα εκτελούνται όταν το σύστημα δέχεται κάποιο (έγκυρο) συνθηματικό χτύπημα. Στη συνέχεια, λοιπόν, θα εξετάσουμε μια κλασική περίπτωση: Θα δούμε πώς πρέπει να ρυθμίσουμε το knockd, ώστε να ανοίγουμε (κι ενδεχομένως να κλείνουμε) το port 22. Ουσιαστικά, το ζητούμενο είναι να επιτρέπουμε την πρόσβαση στην υπηρεσία SSH μόνο σ' εκείνους που γνωρίζουν το συνθηματικό χτύπημα…

Η εύκολη λύση. Το knockd μπορεί να αναγνωρίζει πολλά συνθηματικά χτυπήματα και να προβλέπει για το καθένα μια ξεχωριστή ενέργεια. Οι σχετικές ρυθμίσεις τοποθετούνται στο αρχείο /etc/knockd.conf και για κάθε ζεύγος συνθηματικού-ενέργειας, πρέπει να συντάξουμε κατάλληλα μία μικρή ενότητα. Ας δούμε ένα παράδειγμα:



```
pvar@crunchbang: ~
GNU nano 2.2.6          File: /etc/knockd.conf               Modified

[options]
        UseSyslog

[SSHcontrol]
        sequence    = 6789, 7896, 8967, 9678
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 --j ACCEPT
        tcpflags    = syn
        cmd_timeout = 10
        stop_command= /sbin/iptables -D INPUT -s %IP% -p tcp -dport 22 --j ACCEPT

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Κάθε φορά που το knockd αναγνωρίζει ένα συνθηματικό, μπορεί να εκτελεί όχι μόνο μία, αλλά και δύο εντολές. Στην εικόνα, το knockd έχει ρυθμιστεί να ανοίγει το port22 στο firewall και μετά από δέκα δευτερόλεπτα να το κλείνει. Όσοι προλάβουν, πρόλαβαν! Οι άλλοι θα πρέπει να δώσουν πάλι το συνθηματικό χτύπημα.

1. Αυτό δεν είναι ακριβές αλλά, όπως είπαμε, δεν θα επεκταθούμε στη λογική του iptables.

```
[openSSH]
     sequence  = 6789, 7896, 8967, 9678
     seq_timeout = 5
     command   = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 --j
ACCEPT
     tcpflags  = syn
```

Με την πρώτη γραμμή αποδίδουμε ένα όνομα στην ενότητα που ακολουθεί. Το όνομα δεν επηρεάζει καθόλου του λειτουργία του knock daemon, αλλά χρησιμοποιείται για την καλύτερη οργάνωση του αρχείου ρυθμίσεων. Στη δεύτερη γραμμή περιγράφουμε το συνθηματικό. Για να εκτελεστεί η ενέργεια που θα ορίσουμε αμέσως παρακάτω, ο επισκέπτης θα πρέπει να επιχειρήσει να συνδεθεί στο port 6789, μετά στο 7896, μετά στο 8967 και τέλος στο 9678. Στην επόμενη γραμμή ορίζουμε το χρονικό περιθώριο (σε δευτερόλεπτα) που έχει στη διάθεσή του ο επισκέπτης, για να ολοκληρώσει επιτυχώς την ακολουθία χτυπημάτων. Στη συνέχεια, δίνουμε την εντολή που θέλουμε να εκτελείται όταν το knock daemon εντοπίζει το συνθηματικό χτύπημα που μόλις περιγράψαμε. Στο παράδειγμά μας, η εντολή που ορίσαμε θα προσθέσει στο iptables έναν κανόνα ο οποίος θα επιτρέπει τις συνδέσεις από τη διεύθυνση %IP, στο port 22/TCP. Το "%IP" αποτελεί μία ειδική μεταβλητή του knockd, που αντικαθίσταται αυτόματα από τη διεύθυνση εκείνου που ολοκλήρωσε επιτυχώς το συνθηματικό. Έτσι, το firewall δεν θα ανοίξει το port 22 για όλους, παρά μόνο για εκείνον που έδωσε το σωστό συνθηματικό (ακολουθία χτυπημάτων). Όσο για την τελευταία γραμμή, προσδιορίζει τα "σήματα" που θα πρέπει να διαθέτει μία αίτηση προκειμένου να θεωρείται τμήμα μίας αλληλουχίας χτυπημάτων (ενός συνθηματικού). Σημειώστε ότι το αρχείο ρυθμίσεων του knockd περιλαμβάνει εξαρχής ένα παρόμοιο παράδειγμα. Ωστόσο, καλό θα ήταν να το αντικαταστήσετε με το δικό σας, ώστε να γνωρίζετε το συνθηματικό χτύπημα. Για να ξεκινήσουμε το knockd για πρώτη φορά, πρέπει να επέμβουμε και στο αρχείο /etc/default/knockd. Εκεί, θα βρούμε την παράμετρο START_KNOCKD, στην οποία πρέπει να δώσουμε την τιμή 1:

```
START_KNOCKD=1
```

Αφού αποθηκεύσουμε το αρχείο μετά τη μικρή επέμβαση, μπορούμε να ξεκινήσουμε το knockd:

```
sudo /etc/init.d/knockd start
```

Πλέον, το σύστημα θα εξετάζει τις δικτυακές αιτήσεις που "πέφτουν" πάνω στο κλειστό firewall κι όταν εντοπίζει την αλληλουχία που ορίσαμε θα ανοίγει το port 22!

Η εκνευριστική λύση. Η περίπτωση που εξετάσαμε, αν και μοιάζει αρκετά ασφαλής έχει μία σημαντική αδυναμία: Κάποιος που παρακολουθεί το μηχάνημά μας και sniff-άρει τη δικτυακή κίνηση, θα καταφέρει εύκολα να μάθει το συνθηματικό χτύπημα. Έτσι, πολύ σύντομα, θα μπορεί να ανοίξει μόνος του το port του SSH και να εξαπολύσει, π.χ., μια ωραιότατη επίθεση bruteforce. Μήπως θα έπρεπε να του δυσκολέψουμε λίγο τη ζωή; Μία καλή ιδέα θα ήταν να κλείνουμε το port του SSH αυτόματα, μετά την πάροδο ενός μικρού χρονικού διαστήματος. (Για παράδειγμα, δέκα δευτερόλεπτα αφότου ανοίξει.) Με αυτόν τον τρόπο, εμείς που γνωρίζουμε το username και το password θα προλαβαίνουμε να συνδεθούμε στο μηχάνημα. Οι επιτιθέμενοι, όμως, είναι ζήτημα αν θα προλαβαίνουν να πραγματοποιήσουν δυο-τρεις απόπειρες σύνδεσης. Ο τρόπος για το αυτόματο κλείσιμο είναι πολύ απλός.

```
[SSHcontrol]
    sequence    = 6789, 7896, 8967, 9678
    seq_timeout  = 5
    command     = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 --j
ACCEPT
    tcpflags    = syn
    cmd_timeout  = 10
    stop_command = /sbin/iptables -D INPUT -s %IP% -p tcp -dport 22 --j
ACCEPT
```

Σε αυτή την ενότητα δεν βλέπουμε κάτι καινούργιο, εκτός βέβαια από τις δύο τελευταίες γραμμές. Το cmd_timeout ορίζει το χρόνο (σε δευτερόλεπτα), μετά τον οποίο θα εκτελείται μία δεύτερη εντολή. Αυτή η εντολή προσδιορίζεται στην τελευταία γραμμή, με το stop_command. Η εντολή που φαίνεται στο παράδειγμα διαγράφει από το iptables τον κανόνα που επιτρέπει την πρόσβαση στο port 22. Έτσι, κάθε φορά που κάποιος ολοκληρώνει το συνθηματικό, το knockd ανοίγει το port 22, περιμένει δέκα δευτερόλεπτα και το ξανακλείνει. Πλέον, οι ενοχλητικοί άσχετοι που παίζουν με τις επιθέσεις bruteforce, θα είναι αναγκασμένοι να εκτελούν κάθε λίγο και λιγάκι το συνθηματικό χτύπημα.

Η έξυπνη λύση. Με την προηγούμενη λύση θα σπάσουμε τα νεύρα των επιτιθέμενων, αλλά δεν θα έχουμε απαλλαγεί από κάθε κίνδυνο. Στο κάτω-κάτω, ένας λίγο πιο έξυπνος επιτιθέμενος θα κατασκευάσει ένα ειδικό script, το οποίο σε τακτά χρονικά διαστήματα θα στέλνει αυτόματα το συνθηματικό χτύπημα. Μήπως θα ήταν προτιμότερο να εξασφαλίσουμε ότι το συνθηματικό μας χτύπημα θα μένει κρυφό; Δυστυχώς κάτι τέτοιο είναι πρακτικά αδύνατο, εφόσον ένα απλό sniffing μπορεί εύκολα να αποκαλύψει την αλληλουχία των χτυπημάτων μας. Μπορούμε ωστόσο να κάνουμε κάτι καλύτερο: Να ορίσουμε πολλά συνθηματικά και κάθε φορά που δίνουμε κάποιο εξ αυτών, το knockd να προχωράει στο επόμενο. Για να πετύχουμε κάτι τέτοιο, στο αρχείο ρυθμίσεων του knockd δεν θα ορίσουμε κανένα συνθηματικό. Θα δηλώσουμε όμως ένα αρχείο κειμένου, το οποίο θα περιέχει πολλά συνθηματικά χτυπήματα. Η ενότητα ρυθμίσεων που εξετάσαμε αρχικά θα γίνει κάπως έτσι:

```
[openSSH]
    sequence = /etc/knockd_sequences
    seq_timeout = 5
    command  = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 --j ACCEPT
    tcpflags  = syn
```

Προφανώς, το αρχείο με τα συνθηματικά που καθορίσαμε είναι το /etc/knockd_sequences, αλλά θα μπορούσε να είναι και οποιοδήποτε άλλο μας εξυπηρετεί. Όπως και να 'χει, κάθε συνθηματικό θα πρέπει να βρίσκεται μόνο του, σε μια ξεχωριστή γραμμή και χωρίς "ξένα" στοιχεία. Ένα απόσπασμα του knockd_sequences θα μπορούσε να είναι το ακόλουθο:

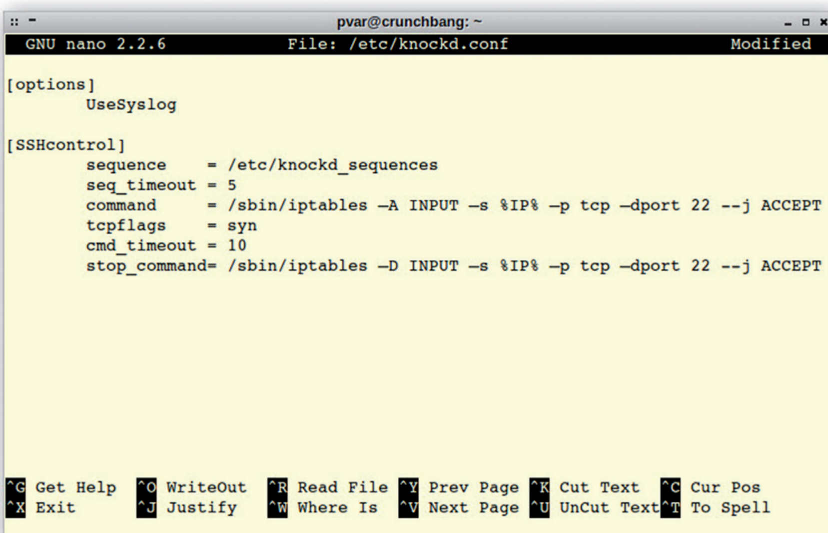
```
...
6789, 7896, 8967, 9678, 4583, 4542, 3599
7481, 9326, 9345, 2593, 4873, 4608, 7451
```

```
9134, 4542, 3599, 8739, 6423, 9535, 5346
5897, 7165, 5531, 4981, 8962, 9125, 1468
...
```

Παραπάνω φαίνονται τέσσερα συνθηματικά, καθένα από τα οποία προβλέπει επτά χτυπήματα. Εδώ πρέπει να σημειώσουμε ότι δεν είναι αναγκαίο για όλα τα συνθηματικά να περιλαμβάνουν το ίδιο πλήθος χτυπημάτων. Επιπρόσθετα, τα χτυπήματα είναι δυνατό να απαιτούν τη χρήση διαφορετικών πρωτοκόλλων και όχι μόνο διαφορετικών ports. Παρακάτω φαίνεται ένα παράδειγμα, ενδεικτικό των ελευθεριών που μόλις αναφέραμε:

```
…
7896:tcp, 8967:udp, 9678:tcp, 4583:udp, 4542:tcp, 3599:tcp
7481:tcp, 9326:udp, 4873:udp, 4608:tcp, 7451:udp
7165:tcp, 5531:udp, 4981:udp, 8962:tcp, 9125:udp, 1468:tcp, 8841:udp
…
```

Είναι αυτονόητο ότι εφόσον ρυθμίσουμε το knockd κατ' αυτόν τον τρόπο, θα πρέπει να εξασφαλίσουμε ότι έχουμε ένα αντίγραφο του knockd_sequences μαζί μας. Επιπρόσθετα, δεν αρκεί να έχουμε τα συνθηματικά μαζί μας. Θα πρέπει και να θυμόμαστε ποιο χρησιμοποιήθηκε την τελευταία φορά. Βλέπετε, κάθε φορά που θα δίνουμε επιτυχώς ένα συνθηματικό, το knockd θα το ξεχνάει και θα μαθαίνει το αμέσως επόμενο… Μπορεί με αυτό το σύστημα να δυσκολεύουμε αρκετά τους επιτιθέμενους, αλλά δυσκολεύουμε και τη δική μας ζωή :/



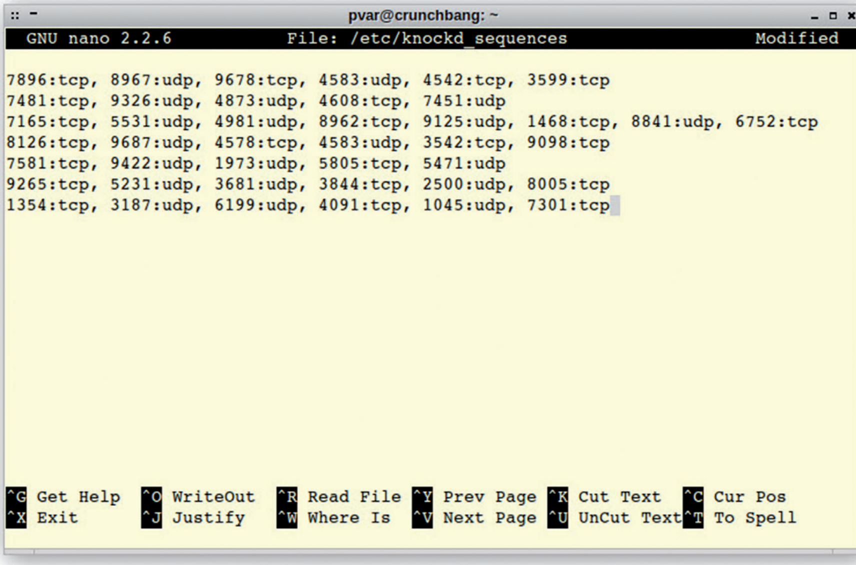
```
pvar@crunchbang: ~
  GNU nano 2.2.6             File: /etc/knockd.conf                   Modified

[options]
        UseSyslog

[SSHcontrol]
        sequence    = /etc/knockd_sequences
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 --j ACCEPT
        tcpflags    = syn
        cmd_timeout = 10
        stop_command= /sbin/iptables -D INPUT -s %IP% -p tcp -dport 22 --j ACCEPT

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Στη θέση του συνθηματικού έχουμε δώσει το όνομα ενός αρχείου. Το knockd θα ανοίξει το συγκεκριμένο αρχείο και θα διαβάσει τα συνθηματικά από εκεί. Κάθε φορά θα απαιτεί να του δίνουμε το επόμενο συνθηματικό που αναφέρεται στο αρχείο.



Στο αρχείο με τα συνθηματικά, κάθε αλληλουχία χτυπημάτων πρέπει να βρίσκεται σε διαφορετική γραμμή.

Στην άλλη όχθη

Μετά από λίγο γράψιμο στο αρχείο ρυθμίσεων του knockd, μπορούμε να το ξεκινήσουμε για μερικές δοκιμές. Ας υποθέσουμε ότι έχουμε πραγματοποιήσει τις ρυθμίσεις που είδαμε στην πρώτη περίπτωση κι ότι το μηχάνημά μας βρίσκεται στη διεύθυνση my_server.org. Όταν βρεθούμε σε κάποιο απομακρυσμένο σύστημα και θελήσουμε να αποκτήσουμε πρόσβαση στην υπηρεσία SSH του server μας, θα πρέπει να δώσουμε κάτι τέτοιο:

```
knock my_server.org 6789 7896 8967 9678
```

Μετά τη διεύθυνση του μηχανήματος, αρκεί να δώσουμε τα ports που πρέπει να "χτυπηθούν". Βέβαια, αν το συνθηματικό απαρτίζεται από χτυπήματα σε ports TCP *και* UDP, θα πρέπει να το λάβουμε κι αυτό υπόψη:

```
knock my_server.org 7481:tcp, 9326:udp, 4873:udp, 4608:tcp, 7451:udp
```

Όπως βλέπετε, η χρήση του knock είναι εξαιρετικά απλή και προφανής! Το μόνο που πρέπει να θυμόμαστε εμείς είναι το συνθηματικό. Αυτό —πιστέψτε μας— θέλει προσοχή! Το ενδεχόμενο να κλειδωθείτε έξω από το μηχάνημά σας είναι πολύ πιθανό.

Στο δεύτερο μέρος, που αρχίζει από τη σελίδα 70, θα μελετήσουμε έναν ακόμα πιο προχωρημένο τρόπο για τη ρύθμιση του knock daemon. Ο στόχος μας είναι να έχουμε διαφορετικό συνθηματικό κάθε φορά, χωρίς να κουβαλάμε μαζί μας μία ολόκληρη λίστα και χωρίς να πρέπει να θυμόμαστε ποιο χρησιμοποιήσαμε κατά την τελευταία σύνδεση. Με αυτή τη μέθοδο, θα σπάνε μόνο τα νεύρα των επιτιθέμενων και όχι τα δικά μας. Για την ώρα, πάντως, σας αφήνουμε να παίξετε με όσα είδαμε και να εξοικειωθείτε με τα βασικά της λειτουργίας του knockd.


Τα επεισόδια του deltaCast, μοιάζουν με ταινίες.
Στη διάρκεια -- όχι στην ποιότητα.
greenqloud
Dashboard
Compute Qloud
Storage Qloud
delta
How to architect your new "new Dropbox"
-- this time around with *really*
enhanced features!
http://deltahacker.gr/deltacast-s02e03

Skill: Intermediate
Tags: Firefox, mobile, development, HTML, JavaScript, CSS

by multiPetros



Ανάπτυξη εφαρμογών για το Firefox OS [Μέρος 1]

Το Firefox OS, το νέο μέλος της οικογένειας προϊόντων του Ιδρύματος Mozilla, έχει σκοπό να μπει για τα καλά στην αγορά των φορητών συσκευών. Πρόκειται για ένα περιβάλλον που ακολουθεί μια διαφορετική προσέγγιση στον τρόπο ανάπτυξης εφαρμογών και παρέχει μεγαλύτερη ελευθερία στον επίδοξο προγραμματιστή. Σε αυτή τη μίνι σειρά άρθρων, θα πραγματοποιήσουμε μια μικρή εισαγωγή στην ανάπτυξη Open Web Apps, όπως ονομάζονται οι εφαρμογές για το Firefox OS.

Οι φορητές συσκευές έχουν κατακλύσει την αγορά και καθορίζουν σε μεγάλο βαθμό τις εξελίξεις στο χώρο της Πληροφορικής. Η εξέλιξη είναι αστραπιαία κι εταιρείες που παραδοσιακά κυριαρχούσαν στο χώρο των PC, με τρανταχτό παράδειγμα τη Microsoft, δεν έχουν καταφέρει να βρουν ακόμα τα πατήματά τους. Από την άλλη, όσες εταιρείες προέβλεψαν την κίνηση της αγοράς, όπως η Apple και η Google, έχουν εξελιχθεί σε κυρίαρχους αυτής της νέας αγοράς.



*Το Alcatel One Touch Fire αποτελεί μια οικονομική πρόταση της Alcatel που φέρει το λειτουργικό Firefox OS και κυκλοφορεί *και* στην ελληνική αγορά.*

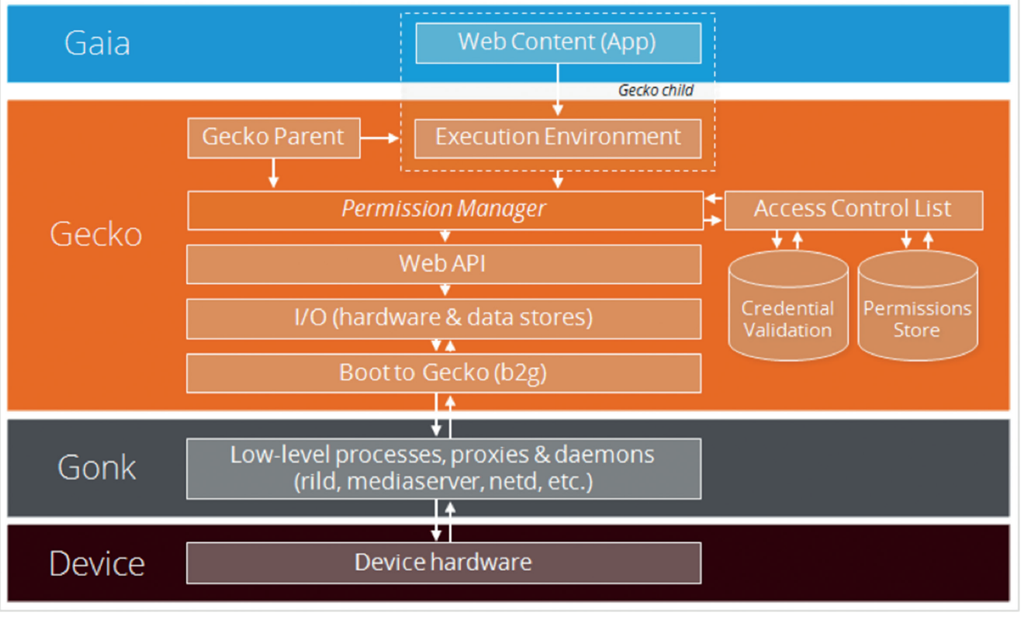
Η μεγάλη δυναμική που παρουσιάζει ο χώρος έχει προσελκύσει το ενδιαφέρον και νέων παικτών, οι οποίοι παρουσιάζουν διάφορες προτάσεις, προσπαθώντας να προσελκύσουν το ενδιαφέρον των προγραμματιστών αλλά και των καταναλωτών. Ορισμένοι από αυτούς, όπως για παράδειγμα η HP με το WebOS, έμειναν στην αφάνεια. Υπάρχουν όμως κι άλλοι, όπως η κοινοπραξία που βρίσκεται πίσω από το Tizen και το ίδρυμα Mozilla με το Firefox OS, που έχουν καταφέρει να μπουν στην αγορά. Μπορεί το "κομμάτι της πίτας" που τους αναλογεί να είναι ακόμα πολύ μικρό, αλλά κανείς δεν μπορεί να αποκλείσει το ενδεχόμενο να μεγαλώσει, καθώς τα προϊόντα του ωριμάζουν. Ιδιαίτερα το Mozilla Foundation έχει πείσει αρκετούς παραδοσιακούς κατασκευαστές, όπως για παράδειγμα την Alcatel, αλλά και νεότερους, όπως την ZTE, να κυκλοφορήσουν συσκευές με Firefox OS. Έτσι, υλοποιήσεις με το λειτουργικό του Mozilla έχουν αρχίσει να φτάνουν ακόμα και στα ράφια των ελληνικών καταστημάτων. Στόχος του ιδρύματος Mozilla είναι η κατασκευή ενός λειτουργικού που θα τρέχει άνετα ακόμα και σε συσκευές low-end. Όπως καταλαβαίνετε, αυτή η αγορά μπορεί να μοιάζει αδιάφορη, αλλά είναι ιδιαίτερα μεγάλη στις αναπτυσσόμενες χώρες και παρουσιάζει ιδιαίτερο οικονομικό ενδιαφέρον.

Όλα αυτά όμως ενδιαφέρουν κυρίως τους αναλυτές της αγοράς, καθώς και όσους θέλουν να επενδύσουν χρόνο και γνώσεις σε μία ανερχόμενη πλατφόρμα. Εμείς αποφασίσαμε να ασχοληθούμε με το Firefox OS για διαφορετικούς λόγους: Επειδή μας αρέσει το μοντέλο ανάπτυξης εφαρμογών που επέλεξε το ίδρυμα Mozilla, αλλά κι επειδή θέλουμε να γνωρίζουμε και να παίζουμε συνεχώς με νέες πλατφόρμες ;) Ας δούμε, λοιπόν, πώς μπορούμε να φτιάξουμε τις δικές μας εφαρμογές για το Firefox OS.

Το λειτουργικό σύστημα

Όπως συμβαίνει με την πλειονότητα των λειτουργικών συστημάτων για φορητές συσκευές, έτσι και το Firefox OS βασίζεται σ' έναν ειδικά διαμορφωμένο πυρήνα Linux. Γύρω απ' αυτόν έχει στηθεί μια μικρή διανομή, προσαρμοσμένη στις ανάγκες των τηλεφώνων και ειδικά εκείνων με "ασθενές" hardware. Το βασικό υπόστρωμα

(software layer) του Firefox OS ονομάζεται Gonk και είναι υπεύθυνο για την επικοινωνία με το hardware της συσκευής. Ένα επίπεδο πιο πάνω βρίσκεται η γνωστή προγραμματιστική μηχανή Gecko. Πρόκειται για τη μηχανή στην οποία στηρίζεται κι ο Firefox. Τέλος, στην κορυφή συναντάμε το υποσύστημα Gaia, που αποτελεί το περιβάλλον χρήσης (User Interface ή UI) του Firefox OS. Το εν λόγω υποσύστημα είναι υπεύθυνο για την αλληλεπίδραση μεταξύ χρήστη και λειτουργικού συστήματος και είναι γραμμένο αποκλειστικά σε HTML, CSS και JavaScript. Με λίγα λόγια, η λειτουργία του στηρίζεται στο Gecko και η επικοινωνία μεταξύ των δύο πραγματοποιείται μέσω web API. Ως αποτέλεσμα, το Gaia είναι αρκετά ελαφρύ και μπορεί να λειτουργήσει σε οποιοδήποτε λειτουργικό σύστημα.



Η αρχιτεκτονική του Firefox OS. Διακρίνονται τα τρία θεμελιώδη software layers: Gonk, Gecko και Gaia.

Ανάπτυξη εφαρμογών

Η ειδοποιός διαφορά του Firefox OS, σε σχέση με τα υπόλοιπα συστήματα των φορητών συσκευών, εντοπίζεται στην ευκολία με την οποία μπορεί κανείς να δημιουργήσει εφαρμογές. Βλέπετε, οι εφαρμογές του Firefox OS αναπτύσσονται σε HTML, CSS και JavaScript, όπως ακριβώς και το Gaia. Ουσιαστικά, οι εφαρμογές του Firefox OS αποτελούν εφαρμογές web, που εκτελούνται με τη βοήθεια της μηχανής Gecko. Ως εκ τούτου, οι προγραμματιστές μπορούν να αναπτύσσουν εφαρμογές χρησιμοποιώντας γνώριμα εργαλεία και τεχνολογίες, ενώ η ευρύτατη γκάμα βιβλιοθηκών JavaScript που υπάρχει ήδη διευκολύνει και επιταχύνει την όλη εργασία.

Φυσικά, οι μηχανικοί του Ιδρύματος Mozilla φρόντισαν να αναπτύξουν το σχετικό WebAPI, ώστε να υποστηρίζει τις λειτουργίες ενός κινητού τηλεφώνου. Μπορείτε να βρείτε πλήρη τεκμηρίωση για τα αντικείμενα και τις συναρτήσεις του WebAPI στη διεύθυνση `http://developer.mozilla.org/docs/WebAPI`. Όπως αντιλαμβάνεστε, καθώς μιλάμε για εφαρμογές web, δεν απαιτείται μεταγλώττιση. Επιπρόσθετα, αν κι ο όρος

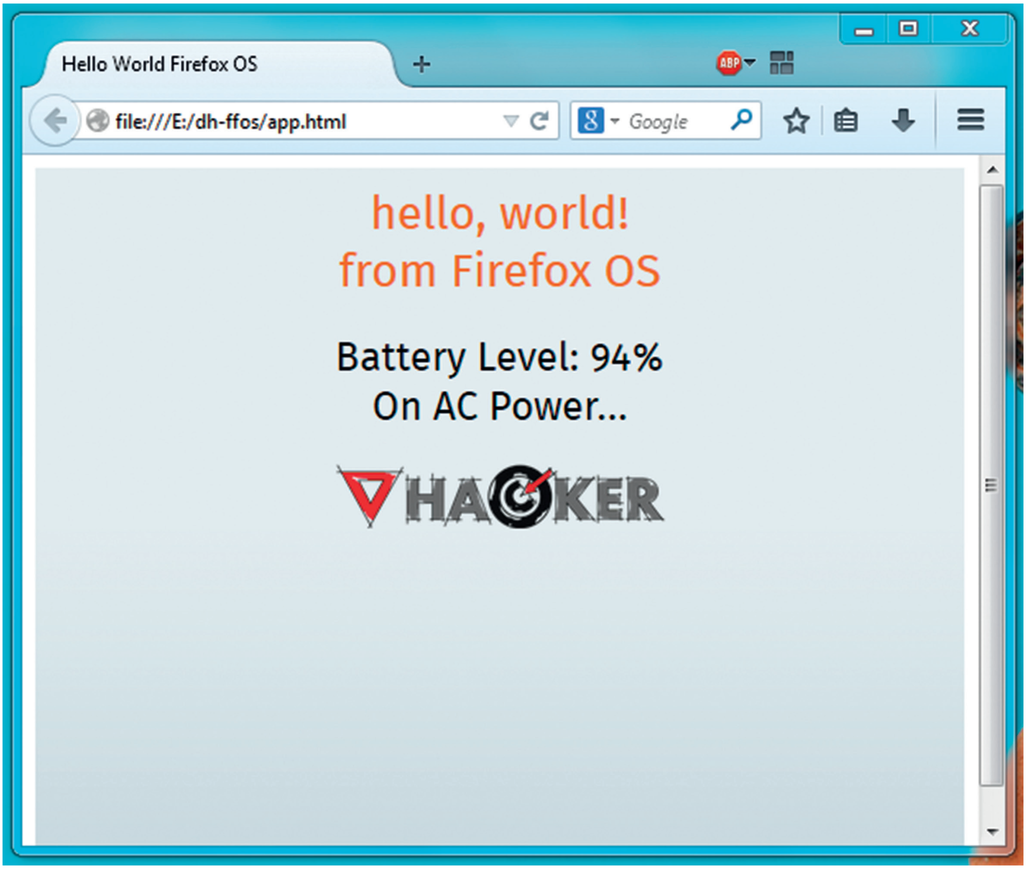
"web" παραπέμπει στο Internet, οι σχετικές εφαρμογές μπορούν να λειτουργούν κι εκτός σύνδεσης (offline).

Για την επίτευξη της ομοιομορφίας των εφαρμογών, η ομάδα ανάπτυξης του Firefox OS δημιούργησε μια σειρά από γραφικά στοιχεία, βελτιστοποιημένα για χρήση σε οθόνες αφής. Τα εν λόγω στοιχεία ονομάζονται "Building Blocks" και μπορείτε να τα κατεβάσετε από το www.buildingfirefoxos.com. Πέρα από αυτά, στην ίδια τοποθεσία θα βρείτε διάφορα σετ εικονιδίων και γραμματοσειρών, καθώς και CSS transitions που χρησιμοποιούνται από το λειτουργικό σύστημα.

Οι εφαρμογές για το Firefox OS έχουν τη γενική ονομασία "open web apps", καθώς στηρίζονται σε ανοιχτά πρότυπα. Όσο για τη διανομή τους, αυτή μπορεί να πραγματοποιηθεί με δύο τρόπους: Είτε με πακέτα ZIP (packaged apps), τα οποία περιλαμβάνουν όλα τα επιμέρους στοιχεία που απαρτίζουν την εφαρμογή (εικόνες, κώδικα HTML, κώδικα JavaScript, style-sheets κ.λπ.), είτε σαν εφαρμογές on-line (hosted apps), των οποίων τα στοιχεία βρίσκονται κάπου στο δίκτυο και μεταφέρονται στη συσκευή του χρήστη *μόνον* όταν πρόκειται να εκτελεστεί η εφαρμογή. Σε κάθε περίπτωση, τα open web applications συνοδεύονται από ένα αρχείο manifest, με το όνομα "manifest.webapp". Σε αυτό το αρχείο περιλαμβάνονται ορισμένες πληροφορίες που απαιτούνται από το λειτουργικό σύστημα, για την ομαλή λειτουργία της αντίστοιχης εφαρμογής. Η δομή των αρχείων manifest στηρίζεται στο πρότυπο JSON, προς μεγάλη χαρά του γράφοντα :D Αν θέλετε να μάθετε περισσότερα για το JSON, μπορείτε να διαβάσετε το άρθρο "Το νέο, βολικό WebStorage και μια ύπουλη αδυναμία", στο deltaHacker 031.

Όπως κάθε λειτουργικό σύστημα για συσκευές mobile, έτσι και το Firefox OS έχει το δικό του "Marketplace", μια τοποθεσία από όπου οι απλοί χρήστες μπορούν να προμηθεύονται εφαρμογές. Η εγγραφή στο Firefox Marketplace είναι πανεύκολη και απολύτως δωρεάν! Πάντως, σε αντίθεση με τους κατασκευαστές των υπολοίπων λειτουργικών, το ίδρυμα Mozilla δεν επιβάλλει την υποχρεωτική χρήση του "επίσημου" Marketplace. Έτσι, οι διάφοροι developers μπορούν να χρησιμοποιούν το "κεντρικό κατάστημα" ή να αναπτύσσουν τα δικά τους μαγαζάκια, από όπου θα διαθέτουν αποκλειστικά τις δικές τους εφαρμογές! Κι όπως ενδέχεται να γνωρίζετε, πέρα από το JSON, ο γράφων αγαπάει και όλες τις λύσεις που είναι self-hosted. Αν όχι, θα ήταν καλή ιδέα να ρίξετε μια ματιά στο άρθρο "Σπιτικό Cloud Storage", στο deltaHacker 031.

Για τη βέλτιστη εμφάνιση των εφαρμογών στο "App Launcher" του λειτουργικού, καθώς και, επί της ουσίας, για τη βέλτιστη εμφάνιση των αντίστοιχων εικονιδίων, η ομάδα του Mozilla παρέχει εκτενείς οδηγίες (http://mozilla.org/styleguide/products/firefox-os/icons). Σε αυτό το σημείο, όμως, είναι πιθανό να σκεφτόσαστε ότι για την ολοκλήρωση μιας εφαρμογής δεν αρκεί η άριστη τεκμηρίωση. Απαιτούνται και πολλές εξαντλητικές δοκιμές! Το ίδρυμα Mozilla έρχεται να μας διευκολύνει *και* σε αυτό το στάδιο. Ένα ειδικό plug-in που έχει δημιουργηθεί για τον Firefox browser, τον μετατρέπει εύκολα κι άμεσα σε ένα simulator του Firefox OS. Μπορείτε να κατεβάσετε το σχετικό plug-in από τη διεύθυνση ftp.mozilla.org/pub/mozilla.org/labs/fxos-simulator. Επιπρόσθετα, όσοι τυγχάνει να κατέχουν ήδη μια συσκευή με το Firefox OS, μπορούν να αποκτήσουν πρόσβαση στην κονσόλα αποσφαλμάτωσης ADB (Android Debug Bridge), είτε με τη βοήθεια ενός κατάλληλου plug-in για τον Firefox (ADB Helper add-on) είτε χρησιμοποιώντας το εργαλείο ADB, στο τερματικό της συσκευής.



Η εφαρμογή μας, όπως δείχνει μέσα από τον Firefox Browser. Παρόλο που δεν βρίσκεται σε περιβάλλον Firefox OS αλλά σε ένα laptop με Windows 8, λειτουργεί άψογα! Βλέπετε, ο αγαπημένος μας browser υποστηρίζει πλήρως το Battery Management API.

Η πρώτη μας εφαρμογή

Ας αφήσουμε όμως τη θεωρία κι ας περάσουμε στην πράξη, γράφοντας την πρώτη μας εφαρμογή για το Firefox OS. Ακολουθώντας την ονοματολογία που έχει επιλέξει το Ίδρυμα Mozilla, θα φτιάξουμε την εφαρμογή "open web". Ως προς τη μορφή της διανομής, επιλέξαμε το μοντέλο του πακέτου ZIP. Αυτό σημαίνει ότι τα αρχεία που απαρτίζουν την εφαρμογή θα αποθηκεύονται στη συσκευή και η εφαρμογή θα λειτουργεί χωρίς να απαιτεί σύνδεση στο δίκτυο. Ας ξεκινήσουμε εξετάζοντας ολόκληρο τον κώδικά της:

```
<html>
 <head>
   <meta charset="UTF-8">
```

```
  <title>Hello World Firefox OS</title>
  <style type="text/css">
  .content {
     padding-top: 10px;
     margin: 0 auto;
     height: 100%;
     background-image: linear-gradient(to bottom, #EAEFF2 50%,
     #D4DDE4 100%);
     font: 24px Fira Sans;
     text-align: center;
  }
  .content h1 {
    font: 28px Fira Sans;
    text-align: center;
    width: 250px;
    margin: 0 auto 20px;
    color: #E66000;
  }
  #logo {
    display: block;
    margin: 20px auto;
    cursor: pointer;
    width: 200px;
  }
  </style>
</head>
<body>
  <article class="content">
    <h1> hello, world!<br>from Firefox OS</h1>
    <div id="battery">
    </div>
    <img id="logo" src="./dh-logo.png" alt="deltaHacker"
    onclick="showMessage()">
  </article>
  <script>
  var battery = navigator.battery ;
```

```

  function updateBatteryStatus(){
    var btLevel = battery.level ;
    var msg = 'Battery Level: ' + Math.round(btLevel * 100) + '%' ;
    if(!battery.charging){
      msg += '<br>Discharging...';
      var dcgTime = battery.dischargingTime ;
      if(isFinite(dcgTime))
        msg += '<br>Remaining time: ' + Math.round(battery.dischargingTime/60) + ' mins' ;
    }
    else{
      msg += '<br>On AC Power...' ;
      var cgTime = battery.chargingTime ;
      if(isFinite(cgTime))
        if(cgTime == 0)
          msg += '<br>Full charged!' ;
        else
          msg += '<br>Fully charged at: ' + Math.round(battery.chargingTime/60) + ' mins' ;
    }
    document.getElementById('battery').innerHTML = msg ;
  }

  battery.addEventListener("chargingchange", updateBatteryStatus) ;
  battery.addEventListener("levelchange", updateBatteryStatus) ;
  updateBatteryStatus() ;

  function showMessage(){
    alert('deltaHacker Rullezzz!') ;
  }

  </script>
 </body>
</html>
```

Ο κώδικας ξεκινά με την περιοχή HEAD, στην οποία περιλαμβάνονται ένας γενικός τίτλος/όνομα της εφαρμογής, η δήλωση του character set που χρησιμοποιείται, καθώς κι ορισμένοι κανόνες για το στυλ. Με λίγα λόγια, ο κώδικας ξεκινά ως μία κλασική σελίδα HTML. Ακολουθεί η περιοχή BODY, με το κυρίως περιεχόμενο της εφαρμογής. Όπως βλέπετε, στη γραμμή 31 έχουμε μια ετικέτα επικεφαλίδας H1, η οποία τυπώνει ένα μήνυμα "hello, world!", ακολουθώντας την παράδοση.

Το μήνυμα "hello, world" προέρχεται από ένα εσωτερικό σημείωμα του Brian Kernighan στα εργαστήρια της Bell Laboratories το 1972, με σκοπό την επίδειξη της χρήσης των εξωτερικών μεταβλητών της γλώσσας B (εξέλιξη της οποίας αποτελεί η γνωστή σε όλους C).

```
main(){
   extrn a,b,c;
   putchar(a); putchar(b); putchar(c); putchar('!*n');
 }
 a 'hell';
 b 'o, w';
 c 'orld';
```

Στη γραμμή 32 ξεκινά μια περιοχή (division), της οποίας το ID φανερώνει ότι θα περιλαμβάνει πληροφορίες σχετικά με τη μπαταρία της συσκευής. Στη γραμμή 34 προστίθεται μια εικόνα, που αποτελεί το λογότυπο του αγαπημένου μας περιοδικού. Αν εξετάσετε τον κώδικα εισαγωγής της εικόνας, θα διαπιστώσετε ότι έχουμε προσθέσει κι έναν event handler, για την περίπτωση που ο χρήστης κάνει κλικ στην εικόνα. Σε αυτή την περίπτωση καλείται η συνάρτηση showMessage(), η οποία εμφανίζει ένα μήνυμα. Το περιεχόμενου του μηνύματος θα το δούμε αργότερα ;)

Κώδικας JavaScript

Στη γραμμή 37 δημιουργούμε τη μεταβλητή battery, στην οποία αποθηκεύουμε ένα αντικείμενο τύπου BatteryManager. Μέσω αυτού του αντικειμένου, το οποίο αποτελεί αντικείμενο του "Battery Status API", αποκτάμε πρόσβαση σε πληροφορίες σχετικά με την κατάσταση της μπαταρίας. Στη γραμμή 39 ξεκινάει η συνάρτηση updateBatteryStatus(), η οποία έχει σαν αποστολή την ενημέρωση της περιοχής (div) που ξεκινούσε στη γραμμή 32 (αυτή με ID to "battery"). Στη γραμμή 40, λοιπόν, δημιουργούμε τη μεταβλητή btLevel, στην οποία εκχωρούμε την τιμή της ιδιότητας level, του αντικειμένου battery. Όπως φαντάζεστε, η ιδιότητα level αποτελεί το επίπεδο φόρτισης της μπαταρίας. Συγκεκριμένα, επιστρέφει μια αριθμητική τιμή τύπου float, στο διάστημα από 0,00 έως 1,00. Στη γραμμή 41 ορίζουμε τη μεταβλητή msg, στην οποία αποθηκεύεται ένα ολοκληρωμένο μήνυμα για την ενημέρωση του χρήστη σχετικά με τη μπαταρία. Για αρχή εκχωρούμε τη συμβολοσειρά "Battery level: " και μετά προσθέτουμε την τιμή που έχει η μεταβλητή btLevel, αφού τη μετατρέψουμε σε ακέραιο αριθμό και συγκεκριμένα σε ένα ποσοστό επί τις εκατό (εδώ αξιοποιούμε τη συνάρτηση Math.round, από τις βιβλιοθήκες της JavaScript). Στην επόμενη γραμμή εκτελούμε έναν έλεγχο της ιδιότητας charging, του αντικειμένου battery. Η συγκεκριμένη ιδιότητα επιστρέφει true εάν η μπαταρία φορτίζεται (αν

είναι συνδεδεμένος ο φορτιστής) και false όταν η μπαταρία αποφορτίζεται. Στην περίπτωση που η μπαταρία αποφορτίζεται, προσθέτουμε στο κείμενο της μεταβλητής msg τη συμβολοσειρά "
Discharging...". Ακολουθεί η δημιουργία της μεταβλητής dcgTime, στην οποία αποθηκεύουμε την τιμή της ιδιότητας dischargingTime. Αυτή η ιδιότητα περιλαμβάνει μια εκτίμηση του χρόνου που απομένει, ως την πλήρη εκφόρτιση της μπαταρίας. Αν είναι αδύνατο να γίνει η σχετική εκτίμηση, λαμβάνουμε μια τιμή infinite. Στη συνέχεια, με τη βοήθεια της συνάρτησης isFinite, ελέγχουμε αν η τιμή του εναπομείναντος χρόνου είναι πεπερασμένη ή όχι. Αν είναι, προσθέτουμε στο μήνυμα τη συμβολοσειρά "
Remaining time: ", ακολουθούμενη από το χρόνο που απομένει σε λεπτά (διαιρούμε την τιμή της dcgTime με το 60 και στρογγυλοποιούμε το αποτέλεσμα).

Στην περίπτωση που η μπαταρία αποφορτίζεται, προσθέτουμε στο μήνυμα τη συμβολοσειρά "
On AC Power..." κι ορίζουμε τη μεταβλητή cgTime. Σε αυτήν δίνουμε την τιμή της ιδιότητας chargingTime, του αντικειμένου battery. Η εν λόγω τιμή αποτελεί μια εκτίμηση του χρόνου που απομένει ως την πλήρη φόρτιση της μπαταρίας. Αν η μπαταρία έχει φορτιστεί πλήρως, η τιμή που λαμβάνουμε είναι μηδενική. Αν πάλι για κάποιο λόγο είναι αδύνατη η εκτίμηση, λαμβάνουμε και πάλι μια τιμή infinite. Αμέσως μετά ελέγχουμε την τιμής της μεταβλητής cgTime. Αν η τιμή της είναι ίση με το 0, προσθέτουμε στο μήνυμά μας το κείμενο "
Full charged!". Διαφορετικά, αν είναι πεπερασμένη και μη-μηδενική, προσθέτουμε το κείμενο "
Fully charged at: ", ακολουθούμενο από τον αριθμό των λεπτών που απομένουν ως την πλήρη φόρτιση (διαιρούμε την cgTime με το 60 και στρογγυλοποιούμε το αποτέλεσμα).

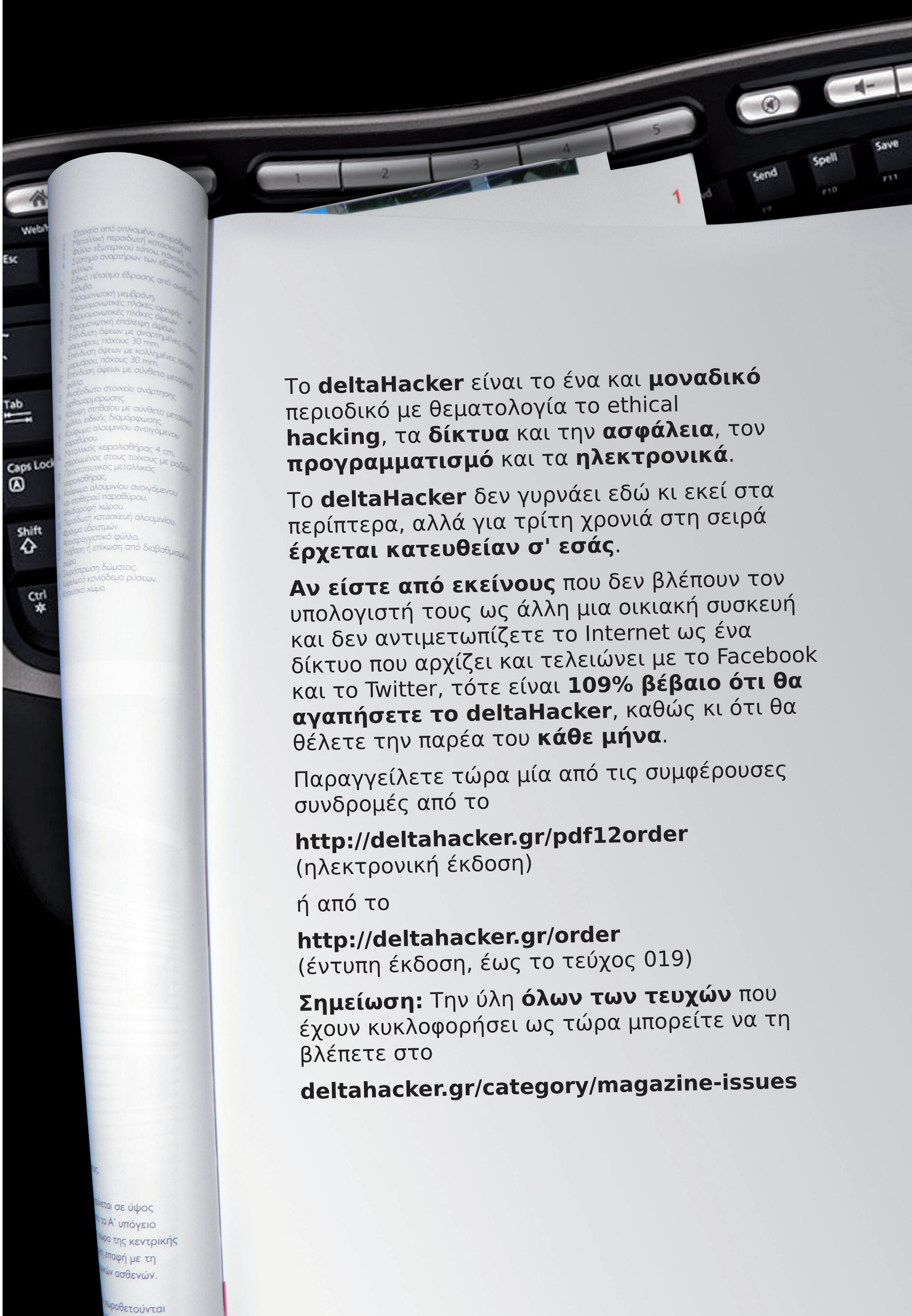
Στο τέλος της συνάρτησης updateBatteryStatus() χρησιμοποιούμε καθαρή JavaScript, ώστε να δώσουμε στην περιοχή (DIV) με ID το "battery" την τιμή της μεταβλητής msg.

Ο κώδικας JavaScript ολοκληρώνεται με δύο event listeners και με μία ακόμα συνάρτηση. Τα event listeners ενεργοποιούνται όταν αλλάξει η κατάσταση της φόρτισης της μπαταρίας (chargingchange) ή το επίπεδο φόρτισης (levelchange). Όταν συμβεί κάτι από τα δύο, καλείται η συνάρτηση updateBatteryStatus(), η οποία τροποποιεί κατάλληλα το εμφανιζόμενο μήνυμα. Η συνάρτηση στο τέλος του κώδικα είναι η showMessage(), η οποία όπως είδαμε καλείται όταν ο χρήστης κάνει κλικ στην εικόνα με το λογότυπο του περιοδικού. Κι όπως βλέπετε η εν λόγω συνάρτηση χρησιμοποιεί τη μέθοδο alert, προκειμένου να εμφανίσει στο χρήστη ένα μήνυμα... αγάπης ;)

Η ώρα της κρίσης...

Είμαστε έτοιμοι. Μόλις φτιάξαμε την πρώτη μας εφαρμογή Open Web! Δεν έχουμε παρά να τη δοκιμάσουμε. Κι επειδή θέλουμε να δούμε αμέσως τη συμπεριφορά της, πριν μπούμε στη διαδικασία να τη φορτώσουμε στον simulator του Firefox OS ή σε κάποια συσκευή, μπορούμε απλά να τη φορτώσουμε στον Firefox browser. Βλέπετε, ο αγαπημένος μας browser υποστηρίζει το "Battery Status API" και η εφαρμογή μας θα λειτουργήσει κανονικά. Έτσι, αν ο υπολογιστής μας είναι laptop, θα πληροφορηθούμε αμέσως για την κατάσταση της μπαταρίας.

Κάπου εδώ θα σας αφήσουμε να κάνετε τις δικές σας δοκιμές, παίζοντας και τροποιώντας τον κώδικά μας. Παρεμπιπτόντως, για δική σας ευκολία, σημειώστε ότι μπορείτε να τον κατεβάσετε από το `http://bit.ly/dH032helloApp` (προσοχή στα πεζοκεφαλαία). Όταν ολοκληρώσετε τους πειραματισμούς σας, μπορείτε να μεταβείτε στη σελίδα 56, όπου έπεται η συνέχεια γύρω από την ανάπτυξη εφαρμογών για το Firefox OS.


Το deltaHacker είναι το ένα και μοναδικό περιοδικό με θεματολογία το ethical hacking, τα δίκτυα και την ασφάλεια, τον προγραμματισμό και τα ηλεκτρονικά.
Το deltaHacker δεν γυρνάει εδώ κι εκεί στα περίπτερα, αλλά για τρίτη χρονιά στη σειρά έρχεται κατευθείαν σ' εσάς.
Αν είστε από εκείνους που δεν βλέπουν τον υπολογιστή τους ως άλλη μια οικιακή συσκευή και δεν αντιμετωπίζετε το Internet ως ένα δίκτυο που αρχίζει και τελειώνει με το Facebook και το Twitter, τότε είναι 109% βέβαιο ότι θα αγαπήσετε το deltaHacker, καθώς κι ότι θα θέλετε την παρέα του κάθε μήνα.
Παραγγείλετε τώρα μία από τις συμφέρουσες συνδρομές από το
http://deltahacker.gr/pdf12order
(ηλεκτρονική έκδοση)
ή από το
http://deltahacker.gr/order
(έντυπη έκδοση, έως το τεύχος 019)
Σημείωση: Την ύλη όλων των τευχών που έχουν κυκλοφορήσει ως τώρα μπορείτε να τη βλέπετε στο
deltahacker.gr/category/magazine-issues

Skill: Intermediate
Tags: stack overflow, buffer overrun, base pointer, stack pointer, return address



by George of the Jungle

Βολικά σφάλματα!

Αναρωτηθήκατε ποτέ πώς καταφέρνουν οι κακόβουλοι hackers να αλλοιώνουν τη συμπεριφορά των προγραμμάτων; Το μυστικό βρίσκεται στην πρόκληση των κατάλληλων σφαλμάτων. Κι επειδή αυτή η δήλωση δεν έχει καμία αξία από μόνη της, αποφασίσαμε να σας το αποδείξουμε. Σε αυτό το άρθρο θα παρουσιάσουμε με τον απλούστερο δυνατό τρόπο μια δημοφιλή τεχνική "επίθεσης", που έχει προκαλέσει πονοκέφαλο σε πολλούς κατασκευαστές προγραμμάτων ;)

Όσοι έχουν έστω και την παραμικρή σχέση με τους υπολογιστές είναι σίγουρο ότι έχουν ακούσει ιστορίες για "φοβερούς και τρομερούς hackers", οι οποίοι προσπέρασαν κάποιο σύστημα ασφάλειας, έσπασαν ένα πρόγραμμα, μπήκαν παράνομα σε κάποιο σύστημα, έκλεψαν προστατευμένα δεδομένα και πάει λέγοντας. Αν και πολλές απ' αυτές τις ιστορίες χαρακτηρίζονται από το στοιχείο της υπερβολής, μη νομίζετε ότι είναι εντελώς φανταστικές. Αρκετές από αυτές είναι αληθινές ή, τέλος πάντων, βασίζονται σε αληθινά γεγονότα. Σκεφτείτε μόνο πόσα προγράμματα κυκλοφορούν σπασμένα ή πόσες επιθέσεις σε δικτυακούς τόπους καταγράφονται καθημερινά. Σκεφτείτε τώρα και τις επιθέσεις που δεν καταγράφονται ή δεν φτάνουν ποτέ στα αφτιά μας. Όλα αυτά όμως τα γράφει κι ο φίλος Jef[1], στο άρθρο του περί προστασίας των εφαρμογών. Εμείς θα κάνουμε το συνήγορο του διαβόλου κι αντί να ασχοληθούμε με την προστασία, θα αναλύσουμε μια δημοφιλή μέθοδο επίθεσης.

Ζήτημα γνώσης...

Οι hackers δεν γνωρίζουν κάποια απόκρυφη τέχνη, ούτε έχουν υπερφυσικές πνευματικές δυνάμεις. Διαθέτουν όμως βαθιές γνώσεις προγραμματισμού, είναι άριστα εξοικειωμένοι με την πλατφόρμα στην οποία δουλεύουν και, πάνω απ' όλα, έχουν καθαρή λογική. Με αυτά τα εφόδια καταφέρνουν, π.χ., να τροποποιούν τη συμπεριφορά μιας εφαρμογής, ώστε να κάνει πράγματα που δεν προβλέπονταν από τον αρχικό προγραμματισμό. Αναρωτιέστε πώς πετυχαίνουν κάτι τέτοιο;

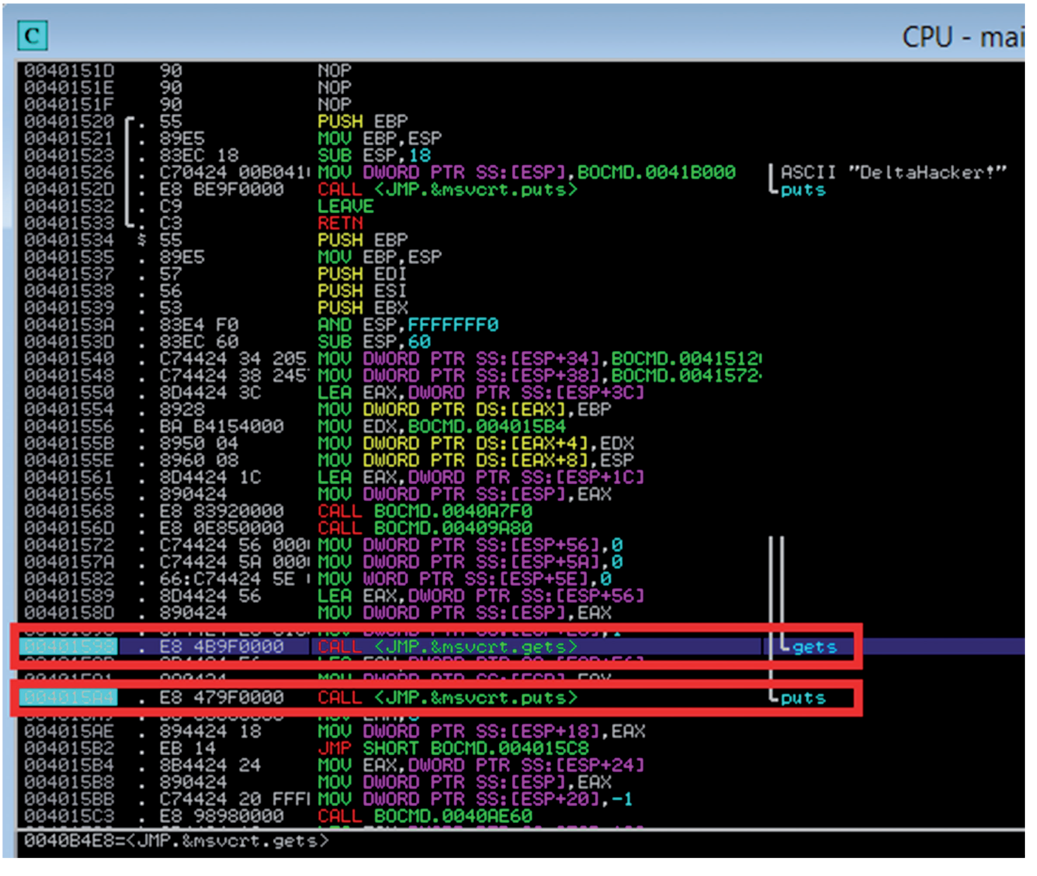
Σε γενικές γραμμές, όταν λέμε ότι κάποιος εκμεταλλεύεται μια αδυναμία εννοούμε ότι εντοπίζει κάποια παράλειψη στον κώδικα, την οποία αξιοποιεί για να προκαλέσει κάποιο σφάλμα στην ομαλή λειτουργία του προγράμματος. Τώρα, όμως, ενδέχεται να σάς δημιουργήσαμε μια νέα απορία: Πώς μπορεί κανείς να προκαλέσει σφάλμα στη λειτουργία ενός προγράμματος; Μια κλασική και απλή περίπτωση, αποτελεί το να δώσουμε σε μια συνάρτηση ή σε κάποια μεταβλητή περισσότερα δεδομένα από αυτά που μπορεί να χειριστεί. Ωραία, θα σκεφτείτε, αφού ένα τόσο απλό σφάλμα είναι στην πραγματικότητα επικίνδυνο, γιατί δεν φροντίζουν οι προγραμματιστές να το αποκλείσουν; Δεν υπάρχουν μέθοδοι για τον έλεγχο των δεδομένων που δίνει ο χρήστης; Αυτές οι ερωτήσεις χτυπάνε φλέβα! Φυσικά και υπάρχουν τέτοιες μέθοδοι, αλλά αυτό δεν σημαίνει ότι τις γνωρίζουν *όλοι* οι προγραμματιστές, ούτε ότι θυμούνται να τις χρησιμοποιήσουν *οποτεδήποτε* κι *οπουδήποτε* απαιτείται. Εξάλλου, ένας προγραμματιστής ενδέχεται να έχει συνεχώς κατά νου τους κινδύνους, αλλά να μην υλοποιεί σωστά τις μεθόδους προστασίας. Μετά από όλα αυτά, ένας νέος στο χώρο του προγραμματισμού και της ασφάλειας είναι πιθανό να σκεφτεί ως εξής: Εντάξει, το να προκαλέσουμε ένα σφάλμα στη λειτουργία ενός προγράμματος δεν είναι αδύνατο. Όμως, πώς είναι δυνατό να προσαρμόσουμε τη συμπεριφορά του στις απαιτήσεις μας, προκαλώντας ένα σφάλμα; Η απάντηση σε αυτό το ερώτημα δεν χωράει σε μία πρόταση. Υπάρχουν πολλές μέθοδοι για την "επίθεση" σε ένα πρόγραμμα και καθεμία θα μπορούσε να αποτελέσει το αντικείμενο ξεχωριστού άρθρου. Γι' αυτό το λόγο θα επικεντρώσουμε μόνο σε μία τεχνική, η οποία θα λέγαμε ότι είναι και διάσημη. Πρόκειται για τη λεγόμενη υπερχείλιση στοίβας, που είναι περισσότερο γνωστή ως stack overflow.

Και τι είναι αυτές οι στοίβες;

Στοίβα (stack) ονομάζεται μια δομή δεδομένων ή, σε απλά ελληνικά, ένας τρόπος διευθέτησης και χειρισμού ενός συνόλου δεδομένων. Οι στοίβες αξιοποιούνται με-

1. Στο τεύχος που αυτή τη στιγμή διαβάζετε ;)

ταξύ άλλων και για τη διαχείριση του προσωρινού αποθηκευτικού χώρου, τον οποίο έχει στη διάθεσή του ένα πρόγραμμα. Το πώς ακριβώς συμπεριφέρονται και ποιες ιδιότητες έχουν, ξεφεύγει από το θέμα μας. Εσείς μπορείτε να φαντάζεστε τη στοίβα σαν το δοχείο που θα χρησιμοποιούσατε για τις βίδες και τα εξαρτήματα, αν αποφασίζατε να ξεβιδώσετε μια περίπλοκη συσκευή. Αυτό το δοχείο θα ήταν χωρισμένο σε μικρές περιοχές, τις οποίες θα αξιοποιούσατε διαδοχικά και με τη σειρά, χωρίς να πηδάτε τυχαία από τη μία θέση στην άλλη και χωρίς να αφήνετε κενά. Αυτό δεν θα το κάνατε μόνο για να αξιοποιήσετε πλήρως το διαθέσιμο χώρο. Θα το κάνατε για να είσαστε σε θέση να αντιστρέψετε τη διαδικασία του ξεβιδώματος και να γνωρίζετε ανά πάσα στιγμή το πού πάει κάθε εξάρτημα και βίδα. Έτσι δεν είναι; Ε, λοιπόν, κάτι αντίστοιχο κάνει και το σύστημα, όταν εκτελεί τον κώδικα ενός προγράμματος. Μόνο που αντί για βίδες κι εξαρτήματα, αποθηκεύει τις τιμές των μεταβλητών και διάφορα άλλα δεδομένα που απαιτούνται για τη μετακίνηση από και προς τις διάφορες συναρτήσεις. Ας αφήσουμε όμως τις αφηρημένες περιγραφές κι ας εξετάσουμε ένα απλό πρόγραμμα σε C.

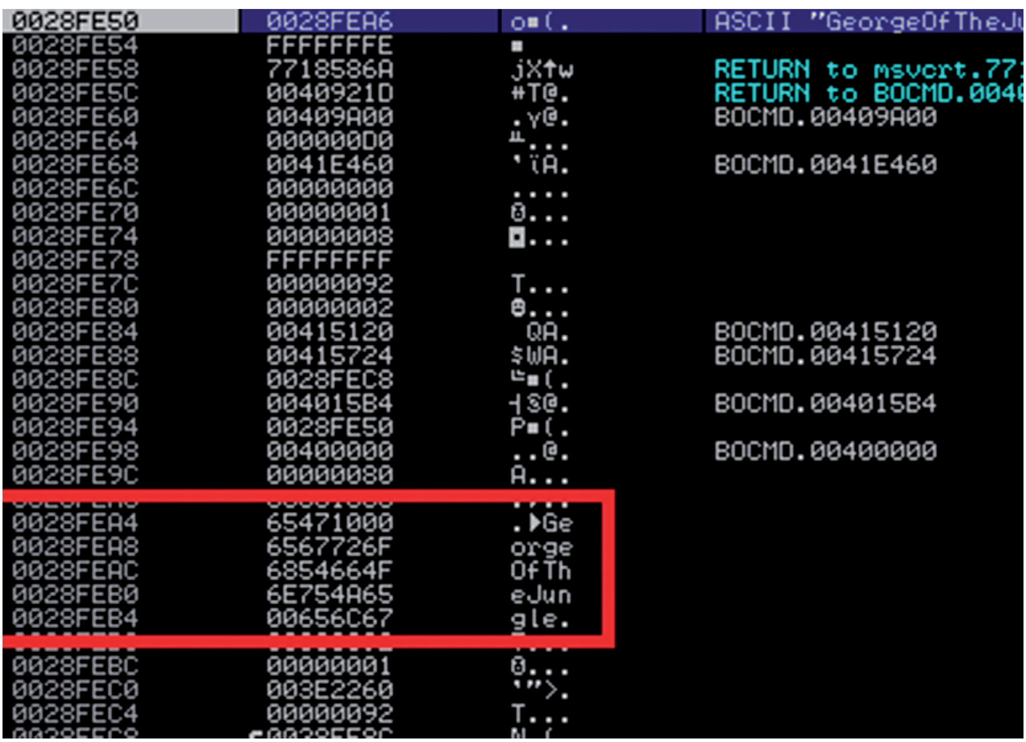


Εικόνα 1
*Εδώ φαίνονται δύο breakpoints: Ένα στην κλήση της συνάρτησης gets και ένα στην κλήση της puts. Για τις ανάγκες της μελέτης μας, πάντως, αρκεί το δεύτερο. Το πρώτο θα σας χρειαστεί μόνον αν θέλετε να εξετάσετε τη στοίβα *πριν* από την εισαγωγή του string…*

```c
#include <stdio.h>
#include <stdlib.h>
void secret_function()  {
  printf("DeltaHacker!\n");
}
void main() {
  char MyBuffer[10]="";
  gets(MyBuffer);
  puts(MyBuffer);
  return 0;
}
```

Όπως βλέπετε, το προγραμματάκι ζητά από το χρήστη να δώσει μια αλληλουχία χαρακτήρων (ένα string) και στη συνέχεια την τυπώνει στην οθόνη. Για να αποκτήσουμε μια εικόνα της μορφής της στοίβας, αρκεί να μεταγλωττίσουμε το παραπάνω πρόγραμμα και να τρέξουμε το εκτελέσιμο μέσα από έναν debugger. Για τις ανάγκες του άρθρου εργαστήκαμε με τον Immunity Debugger, τον οποίο μπορείτε να κατεβάσετε από το www.immunityinc.com/products-immdbg.shtml.

Μετά τη φόρτωση του προγράμματος στον Immunity Debugger, θα δούμε κάτι σαν αυτό που δείχνει η εικόνα 1. Πριν πούμε οτιδήποτε άλλο, έχετε υπόψη ότι αν θέλετε

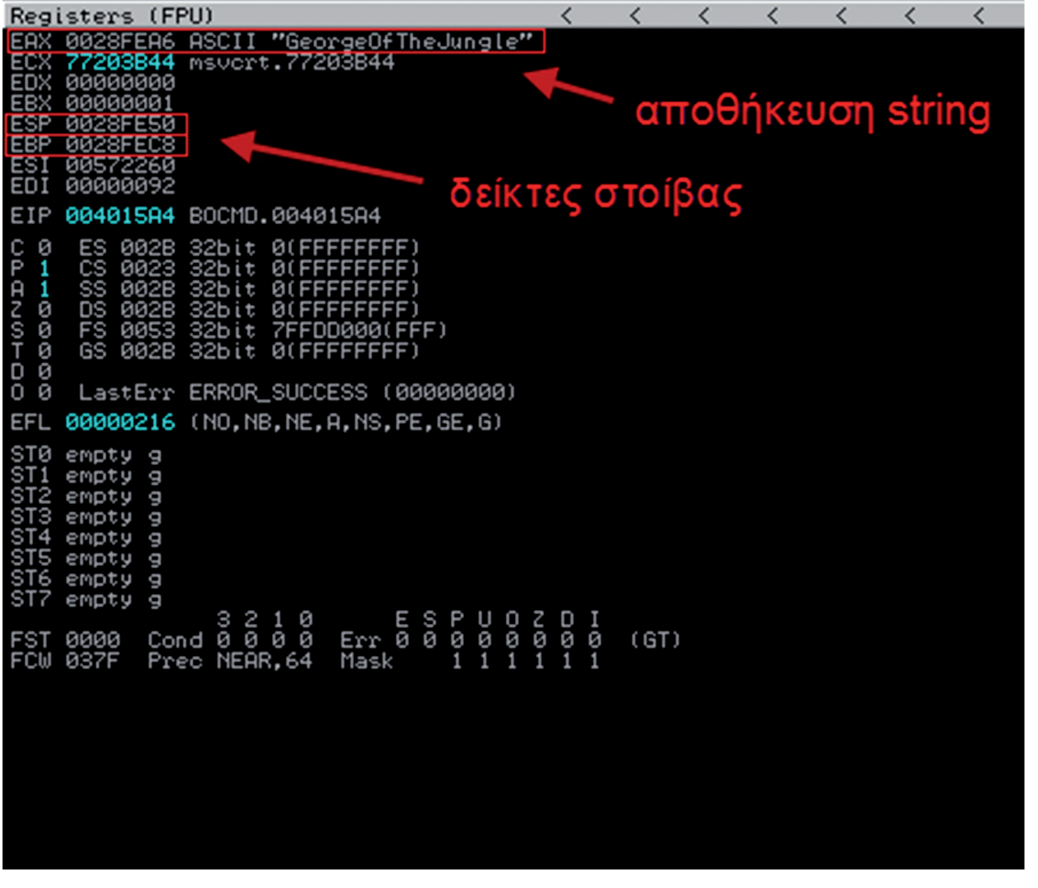


Εικόνα 2
Η στοίβα του προγράμματος, όπως έχει διαμορφωθεί μετά την εισαγωγή του string. Όπως βλέπετε, η αποθήκευση της συμβολοσειράς που δώσαμε ξεκινά στη διεύθυνση 0x28FEA6.

να πραγματοποιήσετε όσα περιγράφονται στη συνέχεια, θα ήταν καλό να αφιερώσετε λίγο χρόνο για τη γνωριμία με το περιβάλλον και το χειρισμό του debugger. Το πρώτο πράγμα που κάναμε εμείς, ήταν να θέσουμε ένα breakpoint στη διεύθυνση 0x4015A4. Δηλαδή στη διεύθυνση από την οποία καλείται η συνάρτηση puts. Αυτό το κάναμε για να διακόψουμε τη ροή του προγράμματος, σε ένα κρίσιμο σημείο για τη μελέτη που ακολουθεί.

Στο εσωτερικό της στοίβας

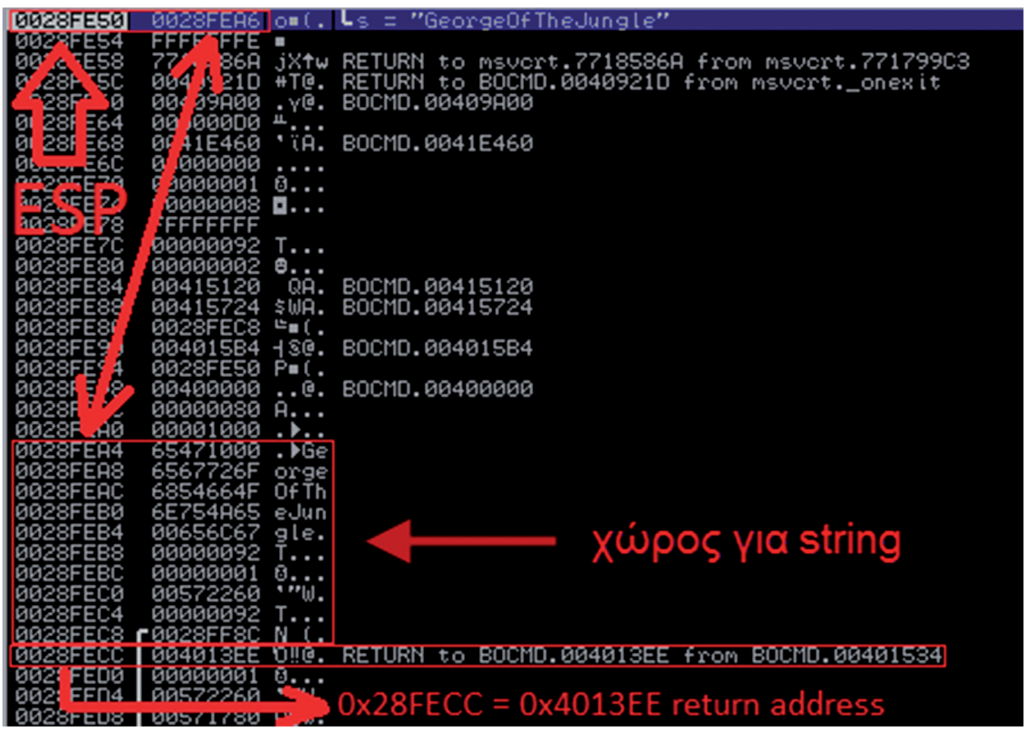
Πατώντας το κουμπί Play ξεκίνησε η εκτέλεση του προγράμματος κι εμείς δώσαμε το string "GeorgeOfTheJungle" (χωρίς τα εισαγωγικά). Όταν όμως πατήσαμε [Enter], η εκτέλεση διακόπηκε σχεδόν ακαριαία. Το πρόγραμμα είχε φτάσει στο breakpoint κι εμείς ήμασταν έτοιμοι να ξεκινήσουμε την εξερεύνηση της στοίβας. Στην εικόνα 2 μπορείτε να δείτε το περιεχόμενό της. Το string που δώσαμε είχε αποθηκευτεί στη διεύθυνση 0x0028FEA6.



Εικόνα 3
Οι πληροφορίες που παρέχει ένας Debugger είναι εξαιρετικά χρήσιμες. Εξετάζοντας την τιμή του καταχωρητή EAX, καταλαβαίνουμε πού τοποθετήθηκε το δοσμένο string. Επιπρόσθετα, από τους καταχωρητές ESP και EBP πληροφορούμαστε για τα όρια της στοίβας.

Στον κώδικα του προγράμματος ορίσαμε έναν πίνακα (μια μεταβλητή τύπου array) με το όνομα Mybuffer. Ως επιθυμητό μέγεθος δηλώσαμε τους 10 χαρακτήρες. Κατά την εκτέλεση του προγράμματος το σύστημα δέσμευσε τον απαιτούμενο χώρο εντός της στοίβας και τοποθέτησε εκεί το string που πληκτρολογήσαμε. Εδώ όμως υπάρχουν αρκετές λεπτομέρειες που αξίζουν την προσοχή μας. Στη συνέχεια θα εξετάσουμε τα όρια της στοίβας, καθώς και την περιοχή που δεσμεύτηκε για το string, αξιοποιώντας τα στοιχεία που παρέχει ο Immunity debugger. Σημειώστε ότι οι εικόνες που ακολουθούν δείχνουν την κατάσταση του προγράμματος κατά την παύση στο breakpoint.

Από την εικόνα 3 διαπιστώνουμε ότι ο καταχωριστής ESP (Stack Pointer) έχει την τιμή 0x28FE50, ενώ ο καταχωριστής EBP (Base Pointer) έχει την τιμή 0x28FEC8. Καλά όλα αυτά, θα πείτε, αλλά που ακριβώς είναι η στοίβα; Ο δείκτης της στοίβας, δηλαδή ο δείκτης του νεότερου στοιχείου της στοίβας, είναι ο ESP. Άρα, το στοιχείο που προστέθηκε τελευταίο στη στοίβα, βρίσκεται στη διεύθυνση 0x28FE50. Η αρχή της στοίβας (η διεύθυνση του πρώτου στοιχείου) ορίζεται από το δείκτη EBP. Επομένως η στοίβα ξεκινά στη διεύθυνση 0x28FEC8. Μη σας κάνει εντύπωση που η αρχή της στοίβας βρίσκεται σε κάποια διεύθυνση *μετά* από το τρέχον στοιχείο και άρα *μετά* από το τέλος της στοίβας. Οι στοίβες διαχειρίζονται το διαθέσιμο χώρο "ανάποδα" αλλά, όπως είπαμε και προηγουμένως, δεν χρειάζεται να μας ανησυχούν



Εικόνα 4

*Από το τέλος του string μέχρι το όριο της στοίβας, φαίνεται να μην υπάρχει τίποτα χρήσιμο. Η αλήθεια είναι ότι αυτό δεν ισχύει, αλλά είναι γεγονός ότι το string θα μπορούσε άνετα να καταλάβει τη συγκεκριμένη περιοχή! Ακριβώς δίπλα βρίσκεται η διεύθυνση επιστροφής, που θα αξιοποιηθεί όταν ολοκληρωθεί η εκτέλεση της gets. Ε, λοιπόν, το string θα μπορούσε να επεκταθεί *και* σε αυτήν την περιοχή. Δεν υπάρχει κανένας μηχανισμός που να ελέγχει το μέγεθός του.*

αυτές οι λεπτομέρειες. Κάτι που έχει περισσότερο ενδιαφέρον είναι το μέγεθος της στοίβας, το οποίο σύμφωνα με τις τιμές των ESP και EBP ανέρχεται στα 120 bytes. Αυτό οφείλεται στο γεγονός ότι η στοίβα αποτελεί έναν προσωρινό αποθηκευτικό χώρο, που δεν προορίζεται αποκλειστικά για την αποθήκευση του string. Θα μπορούσαμε να την παρομοιάσουμε με ένα σημειωματάριο.

Ας εστιάσουμε τώρα στο αποθηκευμένο string. Όπως φαίνεται στην εικόνα 3, η αποθήκευση του κειμένου ξεκινά από τη διεύθυνση 0x28FEA6. Αυτό επαληθεύεται κι από την εικόνα 4, όπου βλέπουμε ότι στη συγκεκριμένη διεύθυνση έχει τοποθετηθεί ο πρώτος χαρακτήρας του string. Πού τελειώνει όμως η περιοχή του string; Όπως είπαμε, ο δείκτης EBP παραπέμπει στην αρχή της στοίβας και η τιμή του είναι 0x28FEC8. Επομένως, η περιοχή του string δεν μπορεί να ξεκινά από εκεί. Ωστόσο, σύμφωνα με τον debugger (δείτε πάλι την εικόνα 4) το αμέσως επόμενο στοιχείο της μνήμης που έχει κάποιο νόημα βρίσκεται στη θέση 0x28FECC. Πρόκειται για μια διεύθυνση επιστροφής και θα ασχοληθούμε μαζί της πολύ σύντομα. Το θέμα είναι ότι από την αρχή του string μέχρι και τη θέση 0x28FECC *φαίνεται* να μην υπάρχει τίποτε άλλο[2]. Κατά συνέπεια, όλος αυτός ο χώρος θα μπορούσε να καταληφθεί από το string. Τελικά, *φαίνεται* ότι για την αποθήκευση του string προσφέρονται 38 bytes.

Στοίβα μέσα στο στοίβα!

Αναφέραμε προηγουμένως ότι το επόμενο στοιχείο της μνήμης, ακριβώς δίπλα στο string, είναι μια διεύθυνση επιστροφής. Για να καταλάβετε τι είναι αυτό και γιατί βρίσκεται σ' αυτή τη θέση, θα χρειαστεί να ανατρέξουμε λίγο στη θεωρία…

Κάθε φορά που καλούμε μια συνάρτηση και πριν ξεκινήσει η εκτέλεσή της, το σύστημα αποθηκεύει τη διεύθυνση της εντολής που βρίσκεται αμέσως μετά την κλήση. Με αυτόν τον τρόπο, όταν ολοκληρωθεί η εκτέλεση της συνάρτησης το σύστημα θα γνωρίζει από πού είχε πραγματοποιηθεί η κλήση και κυρίως από ποιο σημείο του κώδικα πρέπει να συνεχίσει. Με άλλα λόγια, αφού "μεταβεί" στον κώδικα της συνάρτησης και αφού ολοκληρώσει την εκτέλεσή του, θα ξέρει πού να "επιστρέψει". Όπως καταλαβαίνετε, πρόκειται για τη λεγόμενη διεύθυνση επιστροφής (return address). Μήπως αναρωτιέστε πού αποθηκεύεται αυτό το μέγεθος; Μα, εκεί όπου αποθηκεύονται όλες οι προσωρινές πληροφορίες που διατηρεί το πρόγραμμα: Στη στοίβα! Μα καλά, θα πείτε, προηγουμένως είδαμε ότι η διεύθυνση επιστροφής βρίσκεται δίπλα στο string και *έξω* από τη στοίβα. Πώς είναι δυνατόν να λέμε τώρα ότι βρίσκεται *μέσα* στη στοίβα; Η απάντηση είναι απλή, αλλά χρειάζεται λίγη προσοχή: Κάθε φορά που καλείται μία συνάρτηση, το σύστημα δημιουργεί μία στοίβα για την αποθήκευση των σχετικών δεδομένων. Αυτό συνέβη κι όταν ξεκίνησε η συνάρτηση main του προγράμματός μας, οπότε το σύστημα δημιούργησε μια στοίβα. Λίγο αργότερα, όταν κλήθηκε η συνάρτηση gets, το σύστημα δημιούργησε μία πρόσθετη στοίβα. Ε, λοιπόν, αυτή η δεύτερη στοίβα τοποθετήθηκε μέσα στην πρώτη! Για την ακρίβεια, τη στιγμή που κλήθηκε η gets συνέβησαν τα εξής: Το σύστημα αποθήκευσε στη στοίβα της συνάρτησης main τη διεύθυνση επιστροφής, δημιούργησε μια στοίβα για τον κώδικα της συνάρτησης gets και προχώρησε στην εκτέλεση του κώδικα της συνάρτησης.

Ελπίζουμε τα παραπάνω να είναι λίγο-πολύ γνωστά ή, τέλος πάντων, να μη σας

2. Στην πραγματικότητα υπάρχει κάτι, αλλά για τις ανάγκες της παρουσίασής μας μπορούμε να το αγνοήσουμε. Πιστέψτε μας, δεν θα σας βοηθήσει ;)

κούρασαν. Πάντως, από όλα όσα έχουμε πει ως τώρα αρκεί να θυμόσαστε τα εξής: Το string που δώσαμε αποθηκεύτηκε σε μια περιοχή με έκταση 38 bytes. Αυτή η περιοχή βρίσκεται στα όρια της στοίβας που χρησιμοποιείται από τη συνάρτηση gets και, ακριβώς δίπλα, βρίσκεται η διεύθυνση επιστροφής. Με άλλα λόγια, δίπλα στην περιοχή του string βρίσκεται η διεύθυνση μιας εντολής, από την οποία θα συνεχιστεί η εκτέλεση του προγράμματος όταν ολοκληρωθεί η εκτέλεση της συνάρτησης gets.

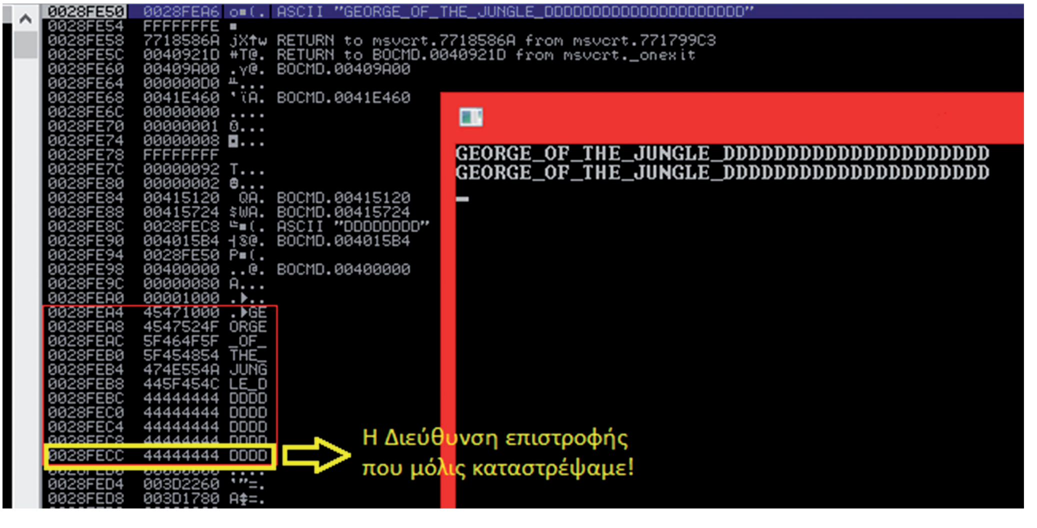
Ισοπεδωτική υπερχείλιση

Πιστεύουμε ότι σ' αυτό σημείο κάποιοι έχουν ήδη αρχίσει να πονηρεύονται: Τι θα γίνει αν πληκτρολογήσουμε ένα string με περισσότερους από 38 χαρακτήρες; Θα προκύψει κάποιο πρόβλημα; Μήπως θα διαμαρτυρηθεί η στοίβα ή η εφαρμογή μας; Η σύντομη απάντηση στα παραπάνω ερωτήματα έχει ως εξής: Θα δημιουργηθεί πρόβλημα, αλλά δεν θα διαμαρτυρηθεί κανένας! Ούτε η στοίβα, ούτε η εφαρμογή, ούτε και κάποιος άλλος. Ας δούμε τώρα τι συνέβη στην πράξη, όταν δώσαμε στο πρόγραμμα ένα τέτοιο string.

Για να τσεκάρουμε τις συνέπειες ξεκινήσαμε την εκτέλεση του προγράμματος από την αρχή και δώσαμε ένα string με 42 χαρακτήρες:

```
GEORGE_OF_THE_JUNGLE_DDDDDDDDDDDDDDDDDDDDDD
```

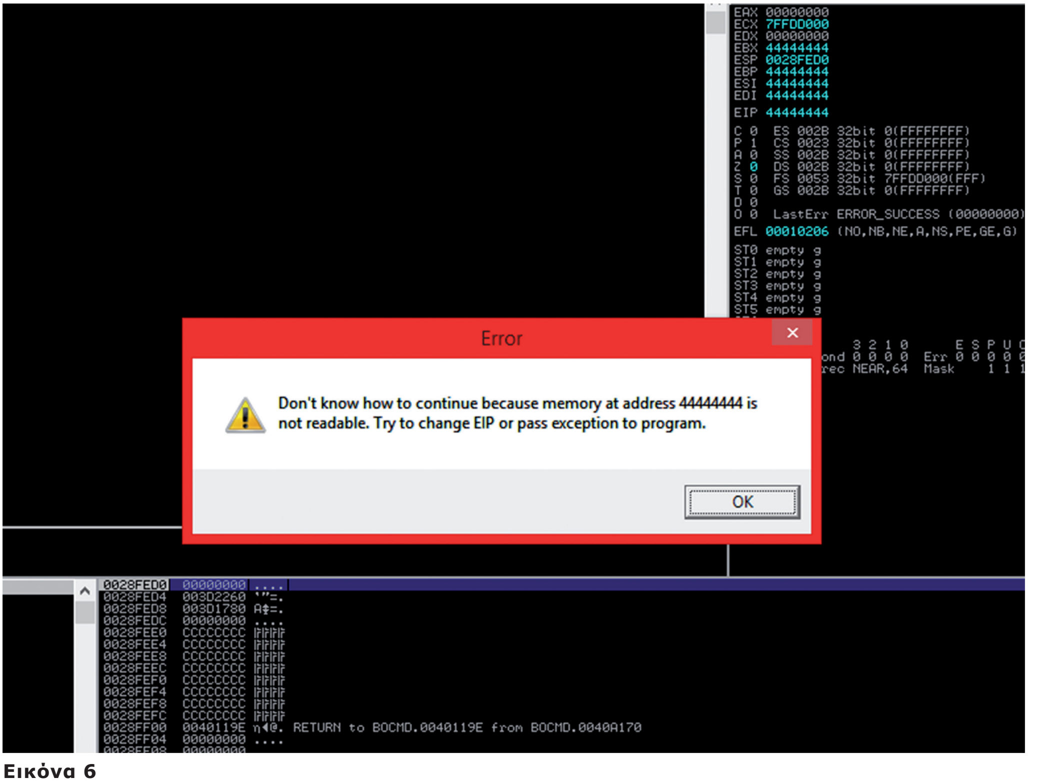
Μπορείτε να φανταστείτε τι έγινε; Η περιοχή της μνήμης που προοριζόταν για το string γέμισε και τα παραπανίσια bytes (τα τελευταία τέσσερα D) τοποθετήθηκαν απ' έξω και, συγκεκριμένα, πάνω στη διεύθυνση επιστροφής. Έτσι, καθώς ο κωδικός ASCII του χαρακτήρα D είναι το 0x44, η διεύθυνση επιστροφής απόκτησε την τιμή 0x44444444. Ουσιαστικά, αυτό που συνέβη ήταν η υπέρβαση των ορίων του διαθέσιμου χώρου για το string κι επειδή αυτός ο χώρος βρισκόταν στα όρια της στοίβας, ξεπεράστηκαν ταυτόχρονα και τα όριά της. Πιο σύντομα, θα μπορούσαμε να πούμε ότι συνέβη ένα buffer overrun, το οποίο οδήγησε και σε ένα stack overflow. (Ψαρωτικό δεν ακούγεται έτσι; :D)



Εικόνα 5

Το μέγεθος του string που πληκτρολογήσαμε ήταν τόσο μεγάλο, που ο διαθέσιμος χώρος για τη στοίβα γέμισε και υπερχείλισε! Οι χαρακτήρες που "ξεχύθηκαν" επικάλυψαν τη διεύθυνση επιστροφής...

Μετά την ολοκλήρωση της συνάρτησης gets το σύστημα προσπάθησε να συνεχίσει την εκτέλεση από εκεί που παρέπεμπε η διεύθυνση επιστροφής (0x44444444). Αυτή η διεύθυνση, όμως, βρισκόταν έξω από την περιοχή μνήμης του προγράμματος. Έτσι, η εκτέλεση του προγράμματος διακόπηκε βίαια. Στην εικόνα 6 μπορείτε να δείτε το σχετικό μήνυμα που εμφάνισε ο debugger.



Εικόνα 6
Μετά την επικάλυψη της διεύθυνσης επιστροφής με τους 4 χαρακτήρες "D", προέκυψε ένα σφάλμα: Το σύστημα αδυνατούσε να βρει τη σχετική διεύθυνση (0x44444444) μέσα στα όρια τις διαθέσιμης μνήμης. Έτσι, το πρόγραμμα τερματίστηκε βίαια :/

Επιτηδευμένη αλλοίωση!

Το σφάλμα που προκαλέσαμε μοιάζει σχεδόν αδιάφορο. Το μόνο που πετύχαμε ήταν ο τερματισμός του προγράμματος. Η αλήθεια όμως είναι ότι πετύχαμε και κάτι πολύ ενδιαφέρον: Αλλοιώσαμε την τιμή της διεύθυνσης επιστροφής. Με την τιμή που της αποδώσαμε δεν συνέβη τίποτα εντυπωσιακό, αλλά αυτό θα μπορούσε να αλλάξει. Αν παρατηρήσετε τον κώδικα του μικρού μας προγράμματος, θα διαπιστώσετε ότι περιλαμβάνει και τη συνάρτηση secret_function, η οποία δεν καλείται από πουθενά. Πώς θα σας φαινόταν αν καταφέρναμε να την εκτελέσουμε, χωρίς να τροποποιήσουμε τον πηγαίο κώδικα του προγράμματος;

Ύπουλος εξαναγκασμός

Εφόσον δεν θα βάλουμε χέρι στον κώδικα του προγράμματος, δεν μπορούμε να καλέσουμε τη συνάρτηση secret_function με τον φυσιολογικό τρόπο, μέσα από τη

συνάρτηση main. Μπορούμε όμως να κάνουμε κάτι πολύ πιο ύπουλο: Μπορούμε να αλλοιώσουμε κατάλληλα το περιεχόμενο της διεύθυνσης επιστροφής, ώστε όταν ολοκληρωθεί η εκτέλεση της gets, το πρόγραμμα να μην επιστρέψει στη συνάρτηση main, αλλά στη συνάρτηση secret_function! Με αυτό το τέχνασμα θα εξαναγκάσουμε το σύστημα να εκτελέσει και μία συνάρτηση η οποία δεν κλήθηκε ποτέ. Ας δούμε τη σχετική διαδικασία, βήμα προς βήμα.

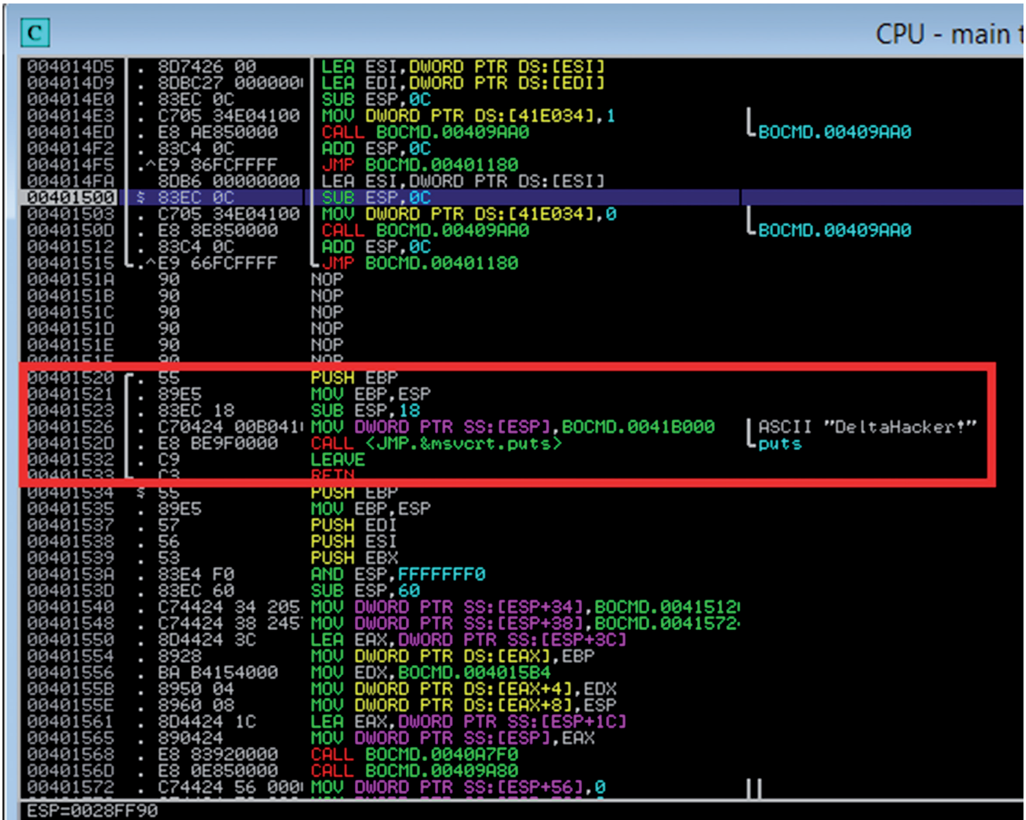
Στο παράδειγμα που εξετάσαμε προηγουμένως, δώσαμε το ακόλουθο string:

```
GEORGE_OF_THE_JUNGLE_DDDDDDDDDDDDDDDDDDDDD
```

Οι χαρακτήρες που σημειώνονται με κόκκινο είναι αυτοί που τίθενται εκτός των ορίων της περιοχής του string και επικαλύπτουν τη διεύθυνση επιστροφής. Επομένως, για να πετύχουμε το στόχο μας, αρκεί να δώσουμε κάτι τέτοιο:

```
GEORGE_OF_THE_JUNGLE_DDDDDDDDDDDDDDDDDδιεύθυνση_της_secret_function
```

Αν ανατρέξουμε στον Immunity Debugger, θα δούμε ότι ο κώδικας της secret_function ξεκινά στη διεύθυνση 0x401520. Ακριβώς εκεί, όπως μπορείτε να δείτε και στην εικόνα 7, βρίσκεται η εντολή PUSH EBP και κατόπιν η MOV EBP, ESP. Με



Εικόνα 7
Εδώ βλέπουμε τα όρια της συνάρτησης secret_function. Η αρχή της σηματοδοτείται από τις εντολές PUSH EBP και MOV EBP,ESP που αρχικοποιούν μια νέα στοίβα, ενώ το τέλος διακρίνεται από την εντολή επιστροφή (RENT).

αυτές τις εντολές, ο δείκτης EBP αποθηκεύεται στη στοίβα και στη συνέχεια παίρνει την τιμή του δείκτη ESP. Επί της ουσίας, εδώ αρχικοποιείται μια νέα στοίβα μέσα στην ήδη υπάρχουσα! Η αρχή της "εσωτερικής" στοίβας τοποθετείται στο τέλος της "εξωτερικής". Όπως είπαμε στη θεωρία, αυτό συμβαίνει κάθε φορά που ξεκινά η εκτέλεση μιας συνάρτησης. Άρα οι παραπάνω εντολές αποτελούν την αρχή κάποιας συνάρτησης. Από τις εντολές που ακολουθούν συμπεραίνουμε ότι αυτή η συνάρτηση είναι όντως η secret_function. Τελικά, τα όρια της secret_function είναι από τη διεύθυνση 0x401520 έως και την 0x401533, όπου συναντάμε την εντολή επιστροφής (RETN από το return).

Εδώ να μπει η εικόνα 007

Ας δούμε τώρα τι θα συνέβαινε δίνοντας το ακόλουθο string:

```
GEORGE_OF_THE_JUNGLE_DDDDDDDDDDDDDDDD \x20\x15\x40\x00
```

Όπως πολύ σωστά φανταστήκατε, η διεύθυνση επιστροφής θα αποκτούσε την τιμή 0x401520. Μήπως σας κάνει εντύπωση που το string περιέχει τα συγκεκριμένα τέσσερα bytes με την ανάποδη σειρά; Αυτό το κάναμε επίτηδες, επειδή γνωρίζουμε ότι θα τοποθετηθούν στη στοίβα ανάποδα! Με αυτό το κολπάκι, όταν ολοκληρωθεί η συνάρτηση gets, το πρόγραμμα θα συνεχιστεί από τη διεύθυνση 0x401520 και ουσιαστικά θα εκτελέσει τη συνάρτηση secret_function!

Η σειρά σας

Κάπου εδώ θα σας αφήσουμε να αφομοιώσετε τη θεωρία που παρουσιάσαμε. Θα σας προτείναμε να δοκιμάσετε και μόνοι όσα είδαμε, τόσο με τον δικό μας κώδικα, όσο και με δικά σας προγραμματάκια που λαμβάνουν δεδομένα από το χρήστη. Κι επειδή ξέρουμε ότι ο πειρασμός είναι μεγάλος, θα σας συμβουλεύαμε να μην πειραματιστείτε με εμπορικές εφαρμογές. Αυτά τα προγράμματα ενσωματώνουν δεκάδες εξελιγμένες μεθόδους προστασίας, τόσο από το stack overflow, όσο και από άλλες επιθέσεις. Επομένως, η μελέτη τους όχι μόνο δεν θα βοηθούσε έναν αρχάριο, αλλά είναι βέβαιο ότι θα προκαλούσε και σύγχυση.



Πριν κλείσουμε, θα σας απευθύνουμε εμείς δύο ερωτήματα: Τι νομίζετε ότι θα συμβεί όταν ολοκληρωθεί η εκτέλεση της secret_function; Μπορείτε να φανταστείτε έναν απλό τρόπο, ώστε να εμποδιστεί το τέχνασμα που παρουσιάσαμε και συνεπώς η παράνομη εγγραφή της διεύθυνσης επιστροφής; Αν πραγματοποιήσετε κι εσείς τα πειράματά μας, θα διαπιστώσετε αμέσως τι συμβαίνει. Μπορείτε όμως να το ερμηνεύσετε; Για το δεύτερο ερώτημα θα σας δώσουμε μια μικρή βοήθεια, με τη μορφή μιας ευχής: Τι καλά που θα ήταν αν η C έλεγχε αυτόματα τα δεδομένα που εισάγει ο χρήστης! Θέλουμε να πιστεύουμε ότι αυτά τα ερωτήματα θα σας πεισμώσουν και θα σας ωθήσουν να μελετήσετε περισσότερο.


Τα επεισόδια του deltaCast, μοιάζουν με ταινίες.
Στη διάρκεια -- όχι στην ποιότητα.
http://deltahacker.gr/deltacast-s02e04

Skill: Intermediate
Tags: Firefox, mobile, development, HTML, JavaScript, CSS

by multiPetros



Ανάπτυξη εφαρμογών για το Firefox OS [Μέρος 2]

Όπως είδαμε στο άρθρο που αρχίζει από τη σελίδα 34, η ανάπτυξη εφαρμογών για το Firefox OS είναι αρκετά εύκολη υπόθεση και δεν διαφέρει ιδιαίτερα από τη δημιουργία μιας σύγχρονης ιστοσελίδας. Σε αυτό το άρθρο θα εξετάσουμε όλες τις απαραίτητες λεπτομέρειες, ώστε η εφαρμογή μας να μεταφερθεί και να λειτουργήσει κανονικά σε μια συσκευή με το Firefox OS -- ή έστω στον σχετικό εξομοιωτή. Επιπρόσθετα, θα εξασφαλίσουμε ότι η εφαρμογή μπορεί να λειτουργεί και εκτός σύνδεσης.

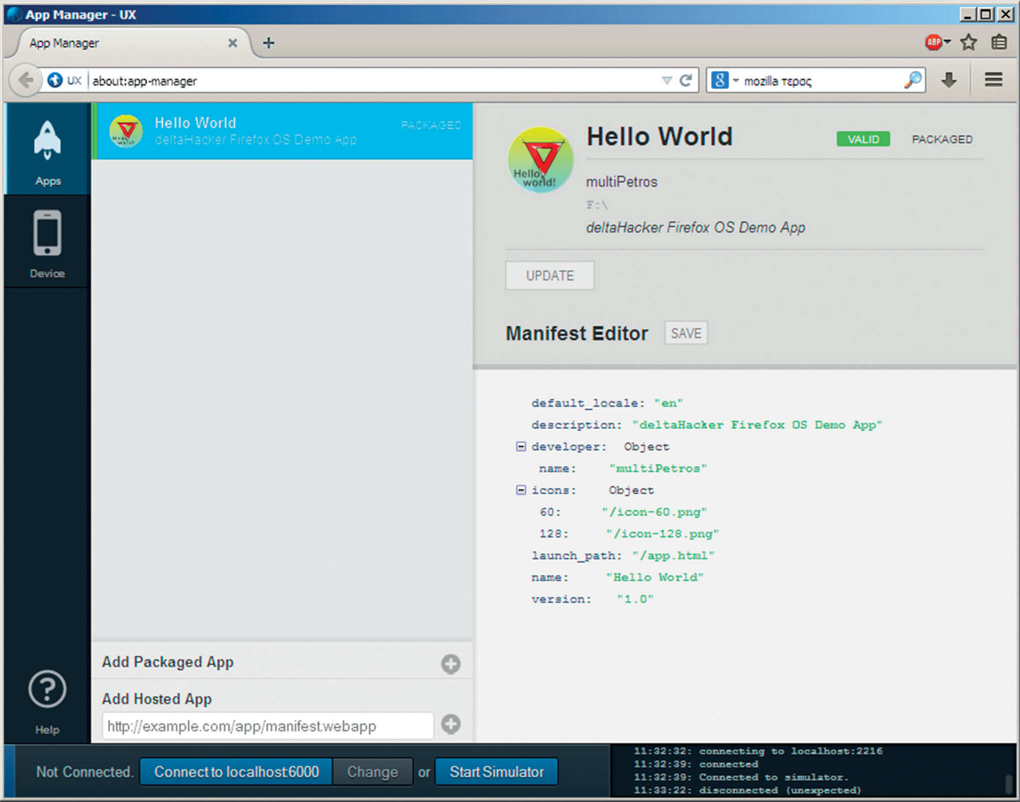
Πριν καταπιαστούμε με την κατασκευή της πρώτης μας εφαρμογής, αναφερθήκαμε στην πλούσια και εκτενή τεκμηρίωση που παρέχει η ομάδα του Ιδρύματος Mozilla. Στις σχετικές σελίδες περιλαμβάνονται έως και οδηγίες για τη σωστή σχεδίαση ενός εικονιδίου. Ξεκινήσαμε από την εν λόγω σελίδα: http://www.mozilla.org/en-US/styleguide/products/firefox-os/icons/. Από εκεί κατεβάσαμε ένα αρχείο PSD, που αποτελεί πρότυπο για τη δημιουργία εικονιδίου. Αφότου ολοκληρώσαμε τις παραμετροποιήσεις και καταλήξαμε στο επιθυμητό αποτέλεσμα, δημιουργήσαμε δύο εικονίδια PNG, το ένα με διαστάσεις 128x128 και το άλλο 60x60 pixel. Πρόκειται για δύο εικονίδια συγκεκριμένων μεγεθών, τα οποία πρέπει να συνοδεύουν κάθε εφαρμογή για το Firefox OS.

Βασικές δηλώσεις

Αυτό που έχει σειρά τώρα είναι το αρχείο manifest.webapp, το οποίο θα πρέπει να συντάξουμε κατάλληλα και να τοποθετήσουμε στη "ρίζα" της εφαρμογής μας: Στον κατάλογο, δηλαδή, που περιλαμβάνει κάθε αρχείο και κάθε άλλο κατάλογο της εφαρμογής. Ας δούμε το περιεχόμενο του αρχείου manifest που γράψαμε για την εφαρμογή μας:

```
{
  "name": "Hello World",
  "description": "deltaHacker Firefox OS Demo App",
  "launch_path": "/app.html",
  "icons": {
  "60": "/icon-60.png",
  "128": "/icon-128.png"
  },
  "version": "1.0",
  "developer": {
    "name": "multiPetros"
  },
  "default_locale": "en"
}
```

Η παράμετρος name, όπως πολύ εύκολα μπορείτε να μαντέψετε, περιέχει το όνομα της εφαρμογής. Η ομάδα του Mozilla προτείνει να μην ξεπερνάμε τους 12 χαρακτήρες, ώστε το όνομα να φαίνεται πάντα ολόκληρο. Αν πάντως θέλουμε να δώσουμε ένα περιγραφικό όνομα στην εφαρμογή μας, δεν θα προκύψει κάποιο σφάλμα. Ακολουθεί η παράμετρος description, στην οποία καταχωρούμε —τι άλλο;— μια περιγραφή της εφαρμογής: Ένα σύντομο κείμενο που αναφέρεται στις βασικές της λειτουργίες. Το launch path προσδιορίζει το αρχείο από το οποίο ξεκινά η εφαρμογή μας, καθώς και τη θέση του σε σχέση με τη ρίζα της εφαρμογής. Συνήθως πρόκειται για ένα αρχείο index.html, αλλά αυτό αποτελεί μια προαιρετική σύμβαση από την οποία



Η καρτέλα του App Manager στον Firefox. Από εδώ μπορούν να φορτωθούν οι Open Web Apps σε μια συσκευή Firefox OS — ή στον αντίστοιχο simulator.

μπορούμε να αποκλίνουμε χωρίς το παραμικρό πρόβλημα. Στην παράμετρο icons καταχωρούνται τα μεγέθη των διαθέσιμων εικονιδίων, καθώς και η θέση του καθενός. Σημειώστε ότι αν σκοπεύετε να εγκαταστήσετε την εφαρμογή σας σε κάποια συσκευή, θα ήταν καλό να έχετε περισσότερα μεγέθη (π.χ., 32x32, 90x90, 120x120 και 256x256).

Πως είπατε; Αναρωτιέστε σε ποια συσκευή θα μπορούσατε να εγκαταστήσετε μια εφαρμογή Open Web; Η απάντηση είναι πολύ πιο απλή απ' όσο φαντάζεστε. Αν έχετε εγκαταστήσει τον Firefox browser σε ένα σύστημα desktop ή σε μια συσκευή με Android, μπορείτε να επισκεφθείτε το Firefox Marketplace κι από εκεί να κατεβάσετε και να εγκαταστήσετε την εφαρμογή σας με ευκολία! Φυσικά, η εκτέλεση της εκάστοτε εφαρμογής θα πραγματοποιείται υπό την αιγίδα της μηχανής Gecko. Και κάπως έτσι, οι εφαρμογές Open Web μπορούν να εισβάλλουν σε κάθε δημοφιλή πλατφόρμα. Ας κλείσουμε όμως αυτή την παρένθεση κι ας επιστρέψουμε στο αρχείο manifest.

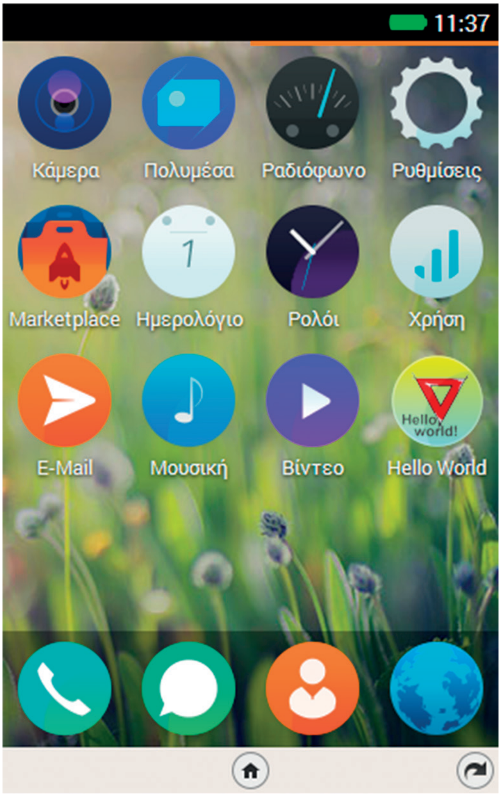
Η επόμενη παράμετρος είναι η version και νομίζουμε ότι δεν χρειάζεται ιδιαίτερες συστάσεις. Το ίδιο ισχύει και για την παράμετρο developer, στην οποία καταχωρούμε το όνομά μας ή το όνομα της εταιρείας μας. Παρεμπιπτόντως, στην παράμετρο developer μπορούμε να προσθέσουμε και ένα πεδίο URL ώστε να δηλώσουμε μια

προσωπική διεύθυνση ή, τέλος πάντων, τη διεύθυνση της εταιρείας μας. Τέλος, με την παράμετρο default_locale ορίζουμε την προεπιλεγμένη γλώσσα της εφαρμογής.

Σε ένα αρχείο manifest μπορούμε να προσθέσουμε πολλές περισσότερες παραμέτρους. Ωστόσο, η απλοϊκή μας εφαρμογή δεν χρειάζεται τίποτα περισσότερο για να λειτουργήσει κανονικά. Για έναν πλήρη κατάλογο των παραμέτρων που μπορούμε να ορίσουμε στο αρχείο manifest, μπορείτε να επισκεφθείτε τη σελίδα `developer.mozilla.org/Apps/Build/Manifest`. Ανάμεσα στις διαθέσιμες παραμέτρους ξεχωρίζει η appcache_path, την οποία θα εξετάσουμε αργότερα.

Δοκιμή των Packaged Apps

Αφού δημιουργήσαμε και το αρχείο manifest, είμαστε έτοιμοι να εγκαταστήσουμε την εφαρμογή μας σε κάποια συσκευή ή στον εξομοιωτή. Εμείς επιλέξαμε να ασχοληθούμε με τη δεύτερη λύση, καθώς πιστεύουμε ότι η διάδοση συσκευών Firefox OS είναι εξαιρετικά περιορισμένη για την ώρα. Ξεκινάμε ανοίγοντας τον Firefox browser και μεταβαίνοντας στο μενού Developer. Εκεί επιλέγουμε το App Manager. Θα εμφανιστεί το παράθυρο διαχείρισης των εφαρμογών, καθώς και μια λίστα με τις συσκευές που τρέχουν Firefox OS κι έχουμε συνδέσει στο σύστημά μας. Εάν δεν διαθέτουμε κάποια (το πιο πιθανό) μπορούμε να τρέξουμε τον εξομοιωτή, κάνοντας κλικ στο Start Simulator κι επιλέγοντας την επιθυμητή έκδοση του Firefox OS. Μέχρι τη στιγμή που γράφονταν αυτές οι γραμμές, η πλέον πρόσφατη αλλά ασταθής (unstable) έκδοση του εξομοιωτή υποστήριζε έως και το Firefox OS 1.5. Πάντως, αν και unstable, ο εξομοιωτής λειτουργούσε κανονικά. Μετά την εκκίνηση του simulator, πατάμε στο κουμπί Add Packaged App. Από το παράθυρο που εμφανίζεται επιλέγουμε τον κατάλογο στον οποίο βρίσκονται τα αρχεία της εφαρμογής μας. Σημειώστε ότι για την προσθήκη στον εξομοιωτή μιας εφαρμογής με τη μορφή "packaged app", δεν χρειάζεται να δημιουργήσουμε κάποιο πακέτο ZIP. Η δημιουργία ZIP απαιτείται όταν σκοπεύουμε να ανεβάσουμε την εφαρμογή μας στο Marketplace. Μόλις επιλέξουμε λοιπόν τον κατάλογο με τα αρχεία της εφαρμογής, αρκεί να πατήσουμε το κουμπί OK. Εκείνη τη στιγμή θα εμφανιστεί στην οθόνη της εικονικής συσκευής ένα παράθυρο, το οποίο θα μας ενημερώνει ότι εγκαταστάθηκε μια νέα εφαρμογή.



Η εφαρμογή "Hello world" εγκαταστάθηκε επιτυχώς στην επιφάνεια εργασίας του Firefox OS! Πλέον, με αυτό το εικονίδιο και μόνο η επιφάνεια εργασίας δείχνει ομορφότερη από ποτέ :P



Εκτελώντας την εφαρμογή "Hello world" λαμβάνουμε τον πολυπόθητο χαιρετισμό, καθώς και πληροφορίες για την κατάσταση της μπαταρίας της συσκευής μας. Σας προτρέπουμε να εγκαταστήσετε κι εσείς την εφαρμογή (έστω και στον simulator) και να πατήσετε στο λογότυπο του περιοδικού... Μπορεί να γνωρίζετε ήδη ποιο μήνυμα θα εμφανιστεί, αλλά πιστεύουμε ότι θα σας χαροποιήσει ούτως ή άλλως :)

Τώρα μπορούμε να περιηγηθούμε στον application launcher του εικονικού συστήματος, όπου θα βρίσκεται πλέον και το εικονίδιο της εφαρμογής μας.

Εφαρμογές hosted

Στην περίπτωση που επιθυμούμε να δημιουργήσουμε μια εφαρμογή hosted —έτσι ονομάζονται οι online εφαρμογές που φιλοξενούνται σε δικτυακό τόπο—, πρέπει να ανεβάσουμε τα σχετικά αρχεία σε κάποιον web server. Για την προσθήκη μιας τέτοιας εφαρμογής στον εξομοιωτή, ανοίγουμε τον Firefox browser, ακολουθούμε τη διαδρομή που περιγράψαμε νωρίτερα και πατάμε το κουμπί Add Hosted App. Στο παράθυρο που εμφανίζεται και στο πεδίο URL, πρέπει να δώσουμε το URL προς το αρχείο manifest της εφαρμογής μας. Ένας σημαντικός περιορισμός που θα πρέπει να έχετε υπόψη αφορά στην υποχρεωτική φιλοξενία μίας και μόνο μίας εφαρμογής ανά subdomain. Αυτό σημαίνει ότι δεν μπορούμε να έχουμε πολλές εφαρμογές σε ξεχωριστούς καταλόγους ενός κοινού subdomain. Για κάθε εφαρμογή μας θα πρέπει να έχουμε δημιουργήσει και ένα αντίστοιχο subdomain! Μια καλή επιλογή για την ονοματολογία, την οποία ακολουθεί και ο γράφων στις δικές του εφαρμογές hosted, είναι η εξής: application_name.mydomain.com.

Μια εφαρμογή hosted έχει όλα τα πλεονεκτήματα των εφαρμογών web, στα οποία συγκαταλέγεται η δυνατότητα πρόσβασης από διαφορετικές συσκευές, η on-line αποθήκευση των δεδομένων, οι δυνατότητες server side scripting καθώς και η μηδενική ανάγκη συντήρησης από πλευράς χρήστη. Βλέπετε, καθώς οι εφαρμογές κατοικούν σε κάποιον web server, ο τελικός χρήστης χρησιμοποιεί πάντα τις πιο πρόσφατες εκδόσεις. Βέβαια όλα αυτά έχουν κι ένα κόστος: Την ανάγκη συνεχούς σύνδεσης στο Διαδίκτυο. Αυτό το εμπόδιο μπορεί να περιοριστεί μερικώς — και σε ορισμένες περιπτώσεις να εξαλειφθεί πλήρως. Για να πετύχουμε κάτι τέτοιο, πρέπει να στραφούμε σε μια τεχνολογία που εισήχθη με την HTML5 και ακούει στο όνομα Application Cache ή, εν συντομία, appcache. Χρησιμοποιώντας την εν λόγω τεχνολογία, δίνουμε στον browser την εντολή να αποθηκεύσει ορισμένα στοιχεία της εφαρμογής όπως, π.χ., βιβλιοθήκες, εικόνες, αλλά κι ολόκληρες σελίδες, σε κάποιον προσωρινό αποθηκευτικό χώρο (δηλαδή στην cache). Έτσι, την επόμενη φορά που θα προσπαθήσουμε να ξεκινήσουμε

την εφαρμογή, τα στοιχεία θα φορτωθούν από τον τοπικό χώρο αποθήκευσης και θα έχουμε μεγάλη εξοικονόμηση στον όγκο των δεδομένων που μεταφορτώνονται. Επιπρόσθετα, αν έχουμε κατεβάσει όλα τα "εξαρτήματα" της εφαρμογής θα μπορούμε να τη χρησιμοποιούμε και offline, σαν να ήταν της μορφής packaged.

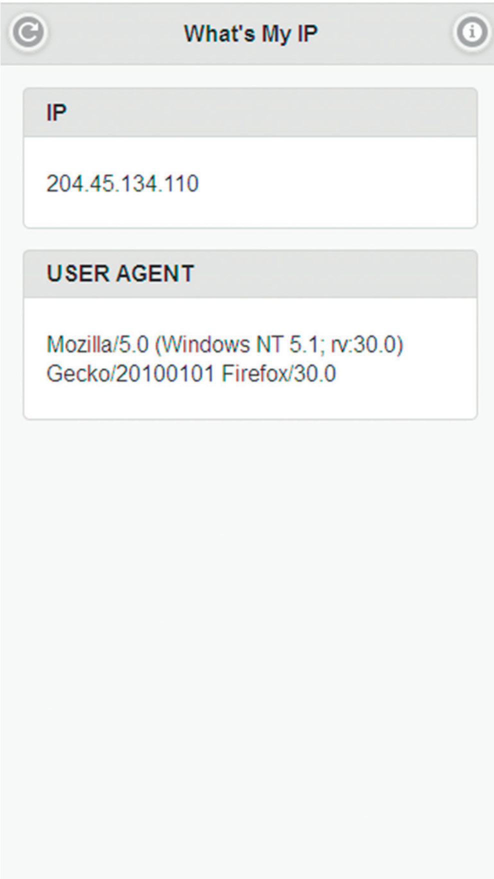
Ρύθμιση του Application Cache

Οι οδηγίες για τη λειτουργία της appcache του browser τοποθετούνται σε ξεχωριστό αρχείο, στο οποίο συνήθως δίνουμε ένα όνομα της μορφής application_namename.appcache. Αυτό το αρχείο πρέπει να δηλώνεται στην ετικέτα HTML της εφαρμογής μας, με την ιδιότητα manifest. Τις σχετικές λεπτομέρειες θα τις δούμε αργότερα, όταν θα εξετάσουμε ένα ολοκληρωμένο παράδειγμα εφαρμογής hosted. Για την ώρα θα μελετήσουμε λίγο τα περιεχόμενα ενός αρχείου .appcache. Πριν όμως κι από αυτό, θα σταθούμε σε μια σημαντική λεπτομέρεια: O server που μοιράζει εφαρμογές hosted, πρέπει να επιστρέφει συγκεκριμένα MIME types για τα αρχεία manifest (αρχεία με κατάληξη .webapp), καθώς και γι' αυτά με τις ρυθμίσεις του appcache (αρχεία με κατάληξη .appcache). Για την ακρίβεια, ο server πρέπει να επιστρέφει τον τύπο application/x-web-app-manifest+json για τα αρχεία manifest και τον τύπο text/cache-manifest για τα αρχεία appcache. Κάτι τέτοιο μπορεί να γίνει πανεύκολα στο apache, εισάγοντας στο αρχείο .htaccess του Document Root τις ακόλουθες γραμμές:

```
AddType application/x-web-app-manifest+json .webapp
AddType text/cache-manifest .appcache
```

Ας δούμε τώρα τη γενική δομή ενός αρχείου appcache.

```
CACHE MANIFEST
# 2014-05-22 – v1.0.1
CACHE:
/favicon.ico
index.html
stylesheet.css
images/logo.png
scripts/main.js
NETWORK:
*
FALLBACK:
/online-page.html /offline-message.html
images/some_images/ images/offline.jpg
```



Η απλουστευμένη έκδοση της εφαρμογής "What's My IP" είναι συμβατή με το Firefox OS, αλλά και με όλα σχεδόν τα λειτουργικά συστήματα. Ας μην ξεχνάμε επίσης ότι μπορεί να λειτουργεί κι εκτός σύνδεσης ;)

Όπως βλέπετε, ένα αρχείο της κατηγορίας ξεκινάει με τη συμβολοσειρά CACHE MANIFEST. Αυτό είναι υποχρεωτικό και δεν αποτελεί προσωπική επιλογή του γράφοντα. Όπως σε πάρα πολλές γλώσσες scripting, ο χαρακτήρας # συμβολίζει την αρχή ενός σχολίου. Έτσι, η δεύτερη γραμμή που φαίνεται στο παράδειγμά μας έχει προστεθεί για την ενημέρωση του αναγνώστη και κυρίως του developer.

Δυο γραμμές χαμηλότερα εμφανίζεται η λέξη CACHE, που σηματοδοτεί την έναρξη μιας ειδικής περιοχής. Τα αρχεία που αναφέρονται εδώ αποθηκεύονται στον προσωρινό αποθηκευτικό χώρο του browser του επισκέπτη, χωρίς να γίνεται κάποιος έλεγχος για την έκδοσή τους (π.χ., για το αν έχουν τροποποιηθεί). Τα συγκεκριμένα αρχεία θα μεταφορτωθούν εκ νέου *μόνον αν* τροποποιηθεί το αρχείο appcache — ή αν ο χρήστης καθαρίσει την cache του browser. Σημειώστε ότι η σελίδα HTML, εντός της οποίας προσδιορίζεται το αρχείο appcache, προστίθεται αυτόματα στην εν λόγω λίστα κι ο browser του επισκέπτη θα την αποθηκεύσει.

Λίγο χαμηλότερα ξεκινά μια άλλη περιοχή, η οποία διακρίνεται από τη λέξη NETWORK. Εδώ ορίζονται τα στοιχεία της εφαρμογής μας που απαιτούν σύνδεση με το δίκτυο. Ο αστερίσκος λειτουργεί σαν μπαλαντέρ και αναφέρεται σε όλα τα URLs του domain, εκτός από αυτά που έχουμε τοποθετήσει στην περιοχή CACHE.

Τέλος, η περιοχή FALLBACK αξιοποιείται όταν εκτελούμε μια εφαρμογή εκτός σύνδεσης. Σε αυτή την περιοχή ορίζονται αντιστοιχίες μεταξύ των αρχείων που βρίσκονται online και κάποιων άλλων που βρίσκονται στη συσκευή του χρήστη. Στο παράδειγμά μας, όταν βρισκόμαστε εκτός σύνδεσης η σελίδα /online-page.html αντικαθίσταται από τη σελίδα /offline-message.html, ενώ όλες οι εικόνες που κανονικά μεταφορτώνονται από τον κατάλογο images/some_images/ αντικαθίστανται από την εικόνα images/offline.jpg.

Και η πίτα ολόκληρη και ο σκύλος χορτάτος!

Καλή η θεωρία, αλλά νομίζουμε ότι έφτασε η ώρα να δούμε και πάλι μια ολοκληρωμένη εφαρμογή, καθώς και τον τρόπο αξιοποίησης της appcache. Για τους σκοπούς του άρθρου, κατασκευάσαμε μια απλοϊκή εκδοχή της εφαρμογής "What's my IP", που μπορείτε να βρείτε στη διεύθυνση myip.apps.multipetros.gr. Η εφαρμογή που κατασκευάσαμε για το άρθρο εμφανίζει τη δημόσια IP της συσκευής μας, καθώς και το user agent string του browser.

Μιας και θέλουμε η εφαρμογή να λειτουργεί σε πολλές συσκευές, προκειμένου να αποφύγουμε τα περίπλοκα ζητήματα συμβατότητας επιλέξαμε να χρησιμοποιήσουμε τη δημοφιλή βιβλιοθήκη jQuery Mobile. Επίσης, όπως αναφέραμε ήδη, θα αξιοποιήσουμε την appcache ώστε ο χρήστης να κατεβάζει ελάχιστα αρχεία όταν χρησιμοποιεί την εφαρμογή, ενώ η ίδια να λειτουργεί κι εκτός σύνδεσης (offline). Με αυτό τον τρόπο η εφαρμογή μας θα βρίσκεται πάντοτε στον application launcher της εκάστοτε συσκευής, όπως άλλωστε και οι εφαρμογές packaged που έχει προσθέσει ο χρήστης.

Ο κώδικας που θα δούμε στη συνέχεια, εκτός απ' όσα έχουμε αναφέρει ως τώρα αξιοποιεί και την PHP. Φυσικά, όπως και με τις εφαρμογές web, ο κώδικας PHP θα εκτελείται μόνο στον server και ποτέ στη συσκευή του χρήστη. Ας ξεκινήσουμε λοιπόν τη μελέτη μας από το αρχείο structure.php. Σε αυτό βρίσκονται δύο συναρτήσεις, οι οποίες δημιουργούν την "επικεφαλίδα" (heading) και το "υποσέλιδο" (footer) των σελίδων που απαρτίζουν την εφαρμογή μας. Οι συγκεκριμένες περιοχές θα έχουν πάντα την ίδια μορφή και το ίδιο περιεχόμενο. Ακριβώς γι' αυτό επιλέξαμε να αναθέσουμε τη δημιουργία τους σε δύο συναρτήσεις: Για να μπορούμε να πραγματοποιούμε αλλαγές, επεμβαίνοντας μόνο σε ένα σημείο του κώδικα.

```
<?php
function print_header($appcache = true) {
?>
<!DOCTYPE html>
<html <?php if($appcache !== false) echo 'manifest="./myip.appcache"' ; ?> lang="el">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <link rel="shortcut icon" href="./favicon.ico" />
  <link rel="apple-touch-icon" href="./ios-icon-144.png" />
  <link rel="apple-touch-icon-precomposed" href="./ios-icon-144.png" />
```

```
  <link href="http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.css" rel="stylesheet" type="text/css">
  <link href="./jquery-custom.css" rel="stylesheet" type="text/css">
  <title>What's My IP</title>
</head>

<body>
<?php
}

function print_footer(){
?>
  <!-- Page: about -->
  <div data-role="page" id="about">
     <div data-role="header">
        <h1>About</h1>
     </div>
     <div role="main" class="ui-content">
        <p>
           <img src="./moz-icon-90.png" style="float: left; padding-right: 1em" />
           dH032 MyIP<br />
           (c) Petros Kyladitis
        </p>
        <p>
           An Open Web Application, targeting Mobile Platforms, to inform users about their public IP, user agent string.
        </p>
     </div>
  </div>
  <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.js"></script>
```

```
43 </body>
44 </html>
45 <?php
46 }
47 ?>
```

Παρατηρείστε ότι η συνάρτηση print_header δέχεται μια παράμετρο, η οποία καθορίζει αν θα ενσωματωθεί το αρχείο appcahe ή όχι. Αυτό το κάναμε διότι, όπως θα δούμε στη συνέχεια, δεν είναι επιθυμητό να αποθηκεύονται όλες οι σελίδες στην cache. Ο σχετικός έλεγχος κι ο καθορισμός του αρχείου appcache πραγματοποιείται στη γραμμή 05.

Σειρά έχει το αρχείο index.php, το οποίο χρησιμοποιεί τις συναρτήσεις του προηγούμενου αρχείου και σχηματίζει μια ολοκληρωμένη σελίδα. Σε αυτήν εμφανίζεται και η δημόσια IP του χρήστη, όπως επίσης και το user-agent string.

```
01 <?php
02 include('structure.php');
03 print_header(false) ;
04 ?>
05  <!-- Page: home -->
06  <div id="home" data-role="page">
07     <div data-role="header">
08        <a href="./index.php"  data-icon="refresh" class="ui-btn-
          left" data-iconpos="notext" data-ajax="false">Refresh</a>
09        <h3>What's My IP</h3>
10        <a href="#about"  data-rel="dialog" data-icon="info"
          class="ui-btn-right" data-transition="pop" data-
          iconpos="notext">About</a>
11     </div>
12     <div role="main" class="ui-content">
13        <div class="ui-corner-all custom-corners">
14           <div class="ui-bar ui-bar-a">
15              <h3>IP</h3>
16           </div>
17           <div class="ui-body ui-body-a">
18              <p><?php echo htmlspecialchars(strip_tags($_
                SERVER['REMOTE_ADDR'])) ; ?></p>
```

```
19          </div>
20        </div>
21        <div class="ui-corner-all custom-corners">
22          <div class="ui-bar ui-bar-a">
23            <h3>USER AGENT</h3>
24          </div>
25          <div class="ui-body ui-body-a">
26            <p><?php echo htmlspecialchars(strip_tags($_
              SERVER['HTTP_USER_AGENT'])) ; ?></p>
27          </div>
28        </div>
29        <iframe src="./appcache.html" width="0" scrolling="no"
          height="0" frameborder="0" seamless="seamless"></iframe>
30      </div>
31    </div>
32  <?php
33  print_footer() ;
34  ?>
```

Ο κώδικας PHP είναι απλούστατος και δεν πιστεύουμε ότι θα σας μπερδέψει. Στις γραμμές 02 και 33 καλούνται οι συναρτήσεις που είδαμε νωρίτερα, για τη δημιουργία της επικεφαλίδας και του υποσέλιδου. Στις γραμμές 18 και 26 τυπώνονται τα ζητούμενα δεδομένα, τα οποία λαμβάνονται από κατάλληλες μεταβλητές του server. Προσέξτε ότι στη γραμμή 02, εκεί όπου καλούμε τη συνάρτηση για την κεφαλίδα, δίνουμε σαν παράμετρο την τιμή false. Με αυτόν τον τρόπο καταδεικνύουμε ότι δεν επιθυμούμε την ενσωμάτωση των οδηγιών appcahe. Θυμόσαστε τι είπαμε στη θεωρία; Η σελίδα που περιλαμβάνει οδηγίες appcache αποθηκεύεται εξ ορισμού! Με το να μην προσθέσουμε αυτές τις οδηγίες στη συγκεκριμένη σελίδα, εξασφαλίζουμε ότι ο browser του χρήστη δεν θα την αποθηκεύει. Στην αντίθετη περίπτωση η εφαρμογή μας θα αχρηστευόταν, αφού ο χρήστης θα έβλεπε πάντα την ίδια εκδοχή της σελίδας και άρα την ίδια διεύθυνση IP και το ίδιο user-agent string.

Ωραία, για να αποφύγουμε τον παραπάνω μπελά, δεν πρέπει να ενσωματώσουμε το αρχείο appcache. Ωστόσο, για να λειτουργεί η εφαρμογή μας *και* εκτός σύνδεσης, χρειαζόμαστε το εν λόγω αρχείο, αφού περιλαμβάνει την αντιστοίχηση μεταξύ των αρχείων που βρίσκονται online κι εκείνων που έχουν αποθηκευτεί τοπικά από τον browser. Ακριβώς γι' αυτό καταφύγαμε σε μια πονηριά. Κοιτάξτε την ετικέτα iframe, στη γραμμή 29. Όπως βλέπετε ορίζουμε κατάλληλα την ιδιότητα src, ώστε να δείχνει στην appcache.html. Ταυτόχρονα, ορίζουμε μηδενικό ύψος και πλάτος για το iframe, καθιστώντας το αόρατο. Μήπως υποψιάζεστε που έγκειται η πονηριά μας; Ας δούμε το περιεχόμενο του appcache.html και θα αποκαλυφθούν όλα:

```
<!DOCTYPE html>
<html manifest="./myip.appcache"><head></head><body></body></html>
```

Τι έχουμε εδώ; Μια κενή σελίδα, η οποία όμως ορίζει τη χρήση της appcache. Με το κολπάκι του αόρατου iframe, εξασφαλίσαμε ότι η appcache θα ενεργοποιηθεί και θα χρησιμοποιηθεί από τον browser του χρήστη *χωρίς* να αποθηκευτεί η κύρια σελίδα, δηλαδή αυτή που εμφανίζει τη διεύθυνση IP και το user-agent string. Ας δούμε τώρα και το περιεχόμενο του αρχείου appcahe:

```
CACHE MANIFEST
# v2
CACHE:
favicon.ico
offline.php
jquery-custom.css
ios-icon-144.png
moz-icon-32.png
moz-icon-60.png
moz-icon-90.png
moz-icon-128.png
http://code.jquery.com/jquery-1.10.2.min.js
http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.js
http://code.jquery.com/mobile/1.4.0/jquery.mobile-1.4.0.min.css
http://code.jquery.com/mobile/1.4.0/images/ajax-loader.gif
FALLBACK:
/index.php /offline.php
/ /offline.php
NETWORK:
*
```

Όπως βλέπετε, επιβάλλουμε την αποθήκευση όλων των εικόνων, των αρχείων της βιβλιοθήκης jQuery mobile και της σελίδας offline.php. Επισης, στην ενότητα fallback δηλώνουμε ότι όταν η σελίδα index.php δεν είναι διαθέσιμη (δηλαδή όταν ο χρήστης βρίσκεται εκτός σύνδεσης) θα πρέπει να χρησιμοποιείται η σελίδα offline.php. Η συγκεκριμένη σελίδα εμφανίζει ένα μήνυμα στο χρήστη, ενώ έχει κι ένα κουμπί reload για την περίπτωση που ο χρήστης τύχει να συνδεθεί εκείνη τη στιγμή. Ας δούμε και τον κώδικα του offline.php:

```
<?php
include('structure.php');
print_header(true) ;
?>
 <!-- Page: home -->
 <div id="home" data-role="page">
    <div data-role="header">
       <h3>What's My IP</h3>
       <a href="#about"  data-rel="dialog" data-icon="info"
       class="ui-btn-right" data-transition="pop" data-
       iconpos="notext">About</a>
    </div>
    <div role="main" class="ui-content">
       <div class="ui-corner-all custom-corners">
          <div class="ui-bar ui-bar-a">
             <h3>No internet connection!</h3>
          </div>
          <div class="ui-body ui-body-a">
             <p>It seems you are not connected to the internet.
             <br>Open your connection and try again...</p>
          </div>
       </div>
       <a data-role="button" data-icon="refresh" href="./index.php"
       data-ajax="false">Reload</a>
 </div>
<?php
print_footer() ;
?>
```

Τέλος, καθώς η εφαρμογή μας είναι Open Web κι επιθυμούμε να την καταστήσουμε εγκαταστήσιμη σε συσκευές με Firefox OS, δημιουργήσαμε και το κατάλληλο αρχείο manifest. Εδώ, η ειδοποιός διαφορά από το αντίστοιχο αρχείο που είδαμε νωρίτερα έγκειται στη χρήση της μεταβλητής appcache_path. Δείτε τον κώδικα και θα καταλάβετε.

```
{
  "name": "dH032 MyIP",
  "version" : "1.0",
  "description": "An Open Web Application, targeting Mobile Platforms,
  to inform users about their public IP and the user agent string",
  "launch_path": "/index.php",
  "appcache_path": "/myip.appcache",
  "icons": {
    "32": "/moz-icon-32.png",
    "60": "/moz-icon-60.png",
    "90": "/moz-icon-90.png",
    "128": "/moz-icon-128.png",
    "144": "/ios-icon-144.png"
  },
  "developer": {
    "name": "Petros Kyladitis",
    "url": "http://www.multipetros.gr"
  },
  "default_locale": "en"
}
```

Η εφαρμογή μας, με δυνατότητα λειτουργίας εκτός σύνδεσης, είναι έτοιμη! Μπορεί να μην πετυχαίνει κάτι σπουδαίο, αλλά συγκεντρώνει ιδιαίτερο ενδιαφέρον. Σκεφτείτε ότι αυτή η εφαρμογή μπορεί να τρέξει σε όλες τις πλατφόρμες στις οποίες τρέχει και ο Firefox browser. Με τη βοήθεια της ομάδας ανάπτυξης του Firefox καταφέραμε να δημιουργήσουμε μια εφαρμογή τόσο για το πολλά υποσχόμενο Firefox OS, όσο και για κάθε άλλο σύστημα! Κάπου εδώ, όμως, θα σας αφήσουμε. Πιστεύουμε ότι σας μεταδώσαμε το μικρόβιο και ότι θα αρχίσετε να ασχολείστε κι εσείς με την ανάπτυξη εφαρμογών Open Web. Τέλος, σημειώστε ότι μπορείτε να κατεβάσετε τον κώδικα που παρουσιάσαμε στο παρόν άρθρο από το http://bit.ly/dH032myIP (το URL είναι case sensitive). Σας ευχόμαστε πολλές και καλές προγραμματιστικές περιπέτειες!

Skill: Intermediate
Tags: port knocking, modulo, passwordless login



port 22

by Spir@lEvolut10n

Αφήστε τους να κρυφακούνε!

Κι αμπάρες βάλαμε στους servers μας, και σύρτες κι απ' όλα. Μέχρι και την πόρτα κρύψαμε, αξιοποιώντας την τεχνική port knocking. Όμως οι παρανοϊκοί με την ασφάλεια δεν ικανοποιούνται ποτέ. Και καλά κάνουν, δηλαδή...

Ωραία η ιδέα του port knocking (βλ. άρθρο σελ. 24). Έρχεται ένας άσχετος έξω από το σύστημά μας, βλέπει τα πάντα κλειστά, απογοητεύεται και φεύγει. Τι πιο απλό; Εδώ που τα λέμε, ακόμα και οι πιο υποψιασμένοι είναι αρκετά πιθανό να αναζητήσουν ευκολότερη λεία. Έτσι, αν όχι τίποτ' άλλο, το σύστημά μας θα ξαλαφρώσει από τις ενοχλητικές επιθέσεις bruteforce. Βέβαια, ο τρόπος σκέψης που μόλις περιγράψαμε είναι αισιόδοξος. Υπάρχει και η άλλη πλευρά του νομίσματος, που δεν είναι τόσο ρόδινη. (Σ.τ.Ε. Άρα: Υπάρχουν όψεις πλευρών νομισμάτων, οι οποίες είναι *τόσο* ρόδινες.) Σύμφωνα με αυτή, οι πιο υποψιασμένοι επιτιθέμενοι δεν θα αποθαρρυνθούν, ούτε θα βαρεθούν. Η υποψία για την παρουσία συστήματος port knocking θα κεντρίσει την περιέργειά τους κι αντί να αδιαφορήσουν για το μηχάνημά μας, θα πέσουν πάνω του με τα μούτρα. Μήπως πρέπει να λάβουμε πρόσθετα μέτρα;

Αλλαγή συνθηματικού

Θα μπορούσε να πει κανείς ότι η βασική αδυναμία του port knocking έχει να κάνει με τη φύση των συνθηματικών χτυπημάτων: Επειδή πρόκειται για διαδοχικές απόπειρες σύνδεσης, είναι πολύ εύκολο για κάποιον να τις καταγράψει και να τις αντιγράψει. Στο κάτω-κάτω, ακόμα κι αν χρησιμοποιούμε κάποιο αρχείο με δεκάδες συνθηματικά χτυπήματα, ένας επίμονος εισβολέας θα μπορούσε να στήσει αφτί για όσο χρειαστεί και να τα καταγράψει όλα. Επομένως, το κόλπο με τα πολλαπλά συνθηματικά δεν θα κρατήσει μακριά έναν αποφασισμένο εισβολέα. Αν θέλουμε να εξασφαλίσουμε ότι το σύστημα port knocking είναι ασφαλές, πρέπει να ορίζουμε νέα συνθηματικά συνεχώς. Όμως αυτό το "συνεχώς" αποτελεί μεγάλο μπελά, αν πρέπει να εργαζόμαστε χειροκίνητα. Στη συνέχεια ωστόσο θα δούμε τις απαραίτητες ρυθμίσεις αλλά κι ένα μικρό script, που θα μας λύσουν τα χέρια.

Αυτόματες ανανεώσεις

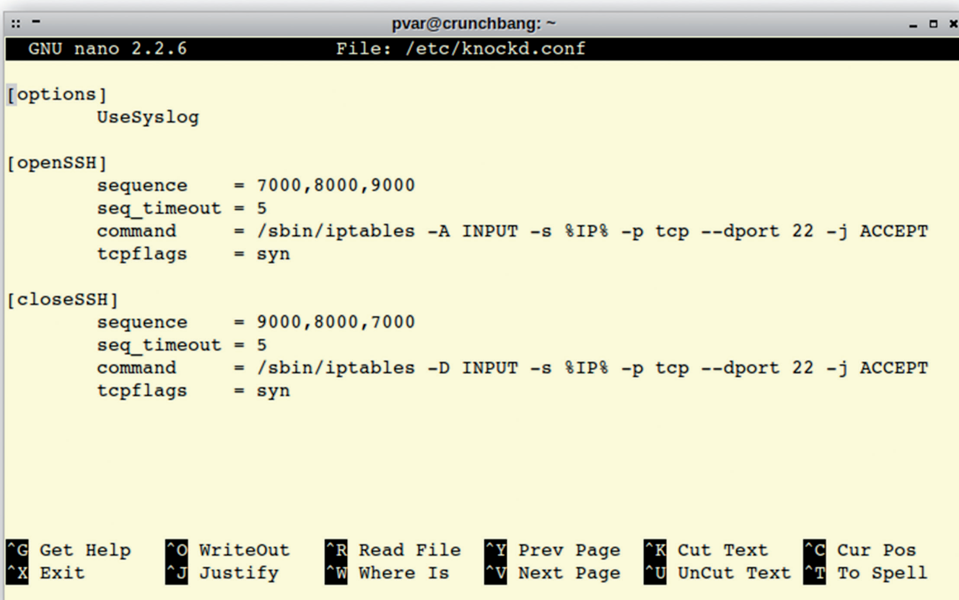
Ας υποθέσουμε ότι έχουμε ένα μηχάνημα που προφυλάσσεται με το σύστημα port knocking. Ας υποθέσουμε, επίσης, ότι θέλουμε να πραγματοποιούμε συνδέσεις προς αυτό, από ένα σύστημα που τρέχει Linux, Unix ή και OS X. Το ιδανικό, από την άποψη της ασφάλειας, θα ήταν να παράγεται και να χρησιμοποιείται ένα νέο συνθηματικό χτύπημα, μετά από κάθε πετυχημένη σύνδεση. Αυτό το συνθηματικό, βέβαια, πρέπει να βρίσκεται στον server με το port knocking αλλά και στο μηχάνημα από το οποίο πραγματοποιούμε τις συνδέσεις. Επομένως, αυτό που θα θέλαμε είναι ένας αυτοματισμός που πραγματοποιεί τα ακόλουθα: Παράγει ένα συνθηματικό μετά από κάθε σύνδεση, ρυθμίζει κατάλληλα το knockd (ώστε να περιμένει πλέον το νέο συνθηματικό) και, τέλος, χρησιμοποιεί κάποια ασφαλή μέθοδο για να στείλει το συνθηματικό στον υπολογιστή μας (από τον οποίο πραγματοποιούμε τις συνδέσεις στον server). Όπως υποψιάζεστε, στις επόμενες ενότητες θα εξετάσουμε μια πιθανή υλοποίηση του παραπάνω μηχανισμού. Η παρουσίασή μας θα ξεκινήσει με τις απαραίτητες εργασίες στον server. Ακολούθως θα προχωρήσουμε στο σύστημα από το οποίο πραγματοποιούνται οι συνδέσεις.

Προετοιμασία

Το πρώτο βήμα είναι να ρυθμίσουμε κατάλληλα το knockd. Για το σκοπό μας, οι εργοστασιακές ρυθμίσεις είναι σχεδόν άριστες. Από μέρους μας απαιτείται μόνο μια μικρή τροποποίηση. Δείτε τις κύριες ενότητες του αρχείου ρυθμίσεων (/etc/knockd):

```
[openSSH]
      one_time_sequences = /home/pvar/.sequence
      seq_timeout = 5
      tcpflags  = syn
      command  = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
[closeSSH]
      sequence = 6009,7008,8007,9006
      seq_timeout = 5
      tcpflags  = syn
      command  = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
```

Όπως παρατηρείτε, έχουμε διατηρήσει τις δύο βασικές λειτουργίες που περιγράφονταν εξαρχής στο αρχείο: Μία για το άνοιγμα της θύρας 22 και μία για το κλείσιμό της. Η αλλαγή που έχουμε κάνει εντοπίζεται στην πρώτη γραμμή της ενότητας "OpenSSH", όπου έχουμε τοποθετήσει την παράμετρο one_time_sequences. Με τη βοήθειά της μπορούμε να ορίζουμε συνθηματικά που θα ισχύουν *μόνο* για μια πετυχημένη σύνδεση το καθένα. Όπως βλέπετε, δεν έχουμε δηλώσει συγκεκριμένα συνθηματικά αλλά ένα αρχείο (/home/pvar/.sequence). Με αυτή τη ρύθμιση δεν λύνουμε το πρόβλημα αλλά το μεταφέρουμε: Πλέον, για την αλλαγή του συνθηματικού δεν απαιτείται να επέμβουμε στο αρχείο ρυθμίσεων του knockd. Αρκεί να "ρίξουμε" το νέο συνθηματικό σ' ένα αρχείο του προσωπικού μας καταλόγου. Για αυτή



```
pvar@crunchbang: ~
  GNU nano 2.2.6              File: /etc/knockd.conf

[options]
        UseSyslog

[openSSH]
        sequence    = 7000,8000,9000
        seq_timeout = 5
        command     = /sbin/iptables -A INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn

[closeSSH]
        sequence    = 9000,8000,7000
        seq_timeout = 5
        command     = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
        tcpflags    = syn

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Το αρχείο ρυθμίσεων του knockd. Για το αυτόματο άνοιγμα του port 22 από το firewall, ο kncokd περιμένει την αλληλουχία χτυπημάτων που βρίσκεται στο αρχείο /home/pvar/.sequence.

τη δουλειά, όπως πολύ σωστά υποψιάζεστε, θα επιστρατεύσουμε ένα script. Πριν καταπιαστούμε με τον κώδικα, θα φροντίσουμε να ξεμπερδέψουμε με το ζήτημα της αυτόματης εκτέλεσής του. Μην ξεχνάτε ότι ο στόχος μας είναι η αυτόματη παραγωγή συνθηματικών, μετά από *κάθε* σύνδεση.

Κάθε φορά που δίνουμε το κατάλληλο συνθηματικό στο σύστημα port knocking, ώστε να ανοίξει το port 22, είναι προφανές ότι σκοπεύουμε να συνδεθούμε μέσω SSH. Όμως, η σύνδεση μέσω SSH συνεπάγεται την εκκίνηση ενός νέου session του BASH. Κάθε session του BASH, με τη σειρά του, συνεπάγεται την αυτόματη εκτέλεση ορισμένων scripts. Επομένως, αν επιθυμούμε την αυτόματη παραγωγή ενός συνθηματικού σε κάθε σύνδεση, θα πρέπει να καλούμε το σκριπτάκι μας από κάποιο αρχείο του BASH. Το εν λόγω αρχείο είναι το .bash_profile και βρίσκεται στον προσωπικό κατάλογο κάθε χρήστη. Παρεμπιπτόντως, αν δεν βρείτε πουθενά το συγκεκριμένο αρχείο (~/.bash_profile), μπορείτε να το δημιουργήσετε δίνοντας κάτι τέτοιο:

```
nano ~/.bash_profile
```

Ορίστε κι ο κώδικας που πρέπει να του προσθέσουμε:

```
if [ -n "${SSH_CLIENT}" ]; then
    ~/bin/new_sequence.sh
fi
```

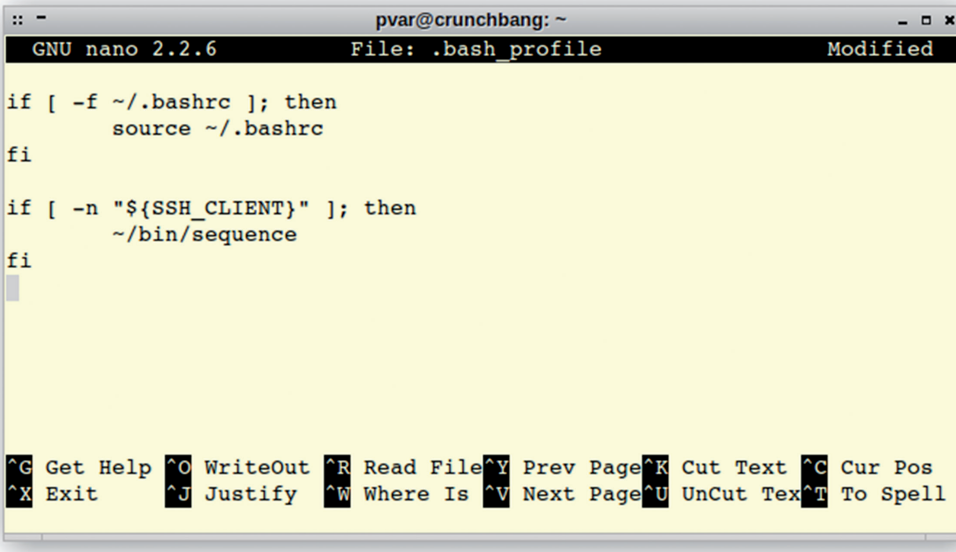
```
pvar@crunchbang: ~
pvar@crunchbang:~$
pvar@crunchbang:~$ sudo iptables -F
pvar@crunchbang:~$ sudo iptables -X
pvar@crunchbang:~$
pvar@crunchbang:~$ sudo iptables -P INPUT DROP
pvar@crunchbang:~$ sudo iptables -P OUTPUT ACCEPT
pvar@crunchbang:~$
pvar@crunchbang:~$ sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
pvar@crunchbang:~$ sudo iptables -A INPUT -i lo -j ACCEPT
pvar@crunchbang:~$
pvar@crunchbang:~$
pvar@crunchbang:~$ sudo iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             state RELATED,ESTABLISHED
ACCEPT     all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
pvar@crunchbang:~$
```

Η ρύθμιση του iptables για τις ανάγκες των δοκιμών μας. Ουσιαστικά, πρόκειται για ρυθμίσεις που αναφέραμε στο πρώτο άρθρο περί port knocking. Όλες οι εισερχόμενες συνδέσεις απορρίπτονται εξ ορισμού, εκτός από όσες σχετίζονται με εξερχόμενες συνδέσεις, καθώς κι όσες προέρχονται από προγράμματα που εκτελούνται τοπικά. Εν ολίγοις, το iptables περιλαμβάνει τα απολύτως απαραίτητα και είναι έτοιμο να δεχτεί "εντολές" από το σύστημα port knocking.

Σ' αυτό το μικρό μπλοκ ελέγχουμε αν το session του BASH ξεκίνησε στο πλαίσιο κάποιας σύνδεσης SSH. Εφόσον συμβαίνει κάτι τέτοιο, ζητάμε από το BASH να εκτελέσει το σκριπτάκι new_sequence, που βρίσκεται στον κατάλογο ~/bin. Αυτό ήταν όλο. Πώς ακριβώς λειτουργεί όμως το new_sequence;



Στο αρχείο .bash_profile του λογαριασμού μας, μπορούμε να προσθέσουμε οποιοδήποτε script θέλουμε να εκτελείται αυτόματα κατά την έναρξη μιας συνεδρίας του BASH. Εμείς τοποθετήσαμε έναν έλεγχο που τσεκάρει αν η σύνδεση έγινε μέσω SSH. Σε αυτή την περίπτωση, εκτελείται το σκριπτάκι που παράγει το νέο συνθηματικό για το port knocking.

Δημιουργία συνθηματικού

Καταρχάς, έχετε υπόψη ότι μπορείτε να κατεβάσετε το script για την παραγωγή των συνθηματικών από το http://bit.ly/dh032newseq. Το σκριπτάκι παράγει μια αλληλουχία τυχαίων αριθμών, που χρησιμοποιείται σαν το νέο συνθηματικό για το port knocking. Αυτή η αλληλουχία τοποθετείται στο αρχείο ~/.sequence, ενώ αντιγράφεται και σ' ένα ομώνυμο αρχείο, στο μηχάνημα από το οποίο πραγματοποιούμε τις συνδέσεις. Για την αντιγραφή χρησιμοποιείται το scp (secure copy), ώστε τα δεδομένα να ταξιδεύουν κρυπτογραφημένα. Στη συνέχεια θα εξετάσουμε τον κώδικα τμηματικά.

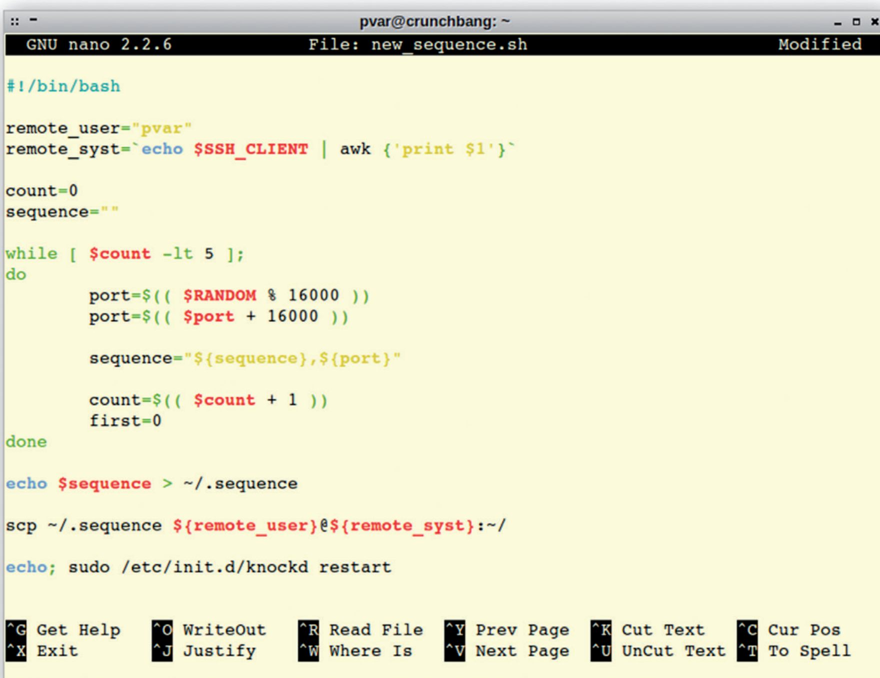
```
#!/bin/bash
remote_user="pvar"
remote_syst=`echo $SSH_CLIENT | awk {'print $1'}`
count=0
sequence=""
```

Αφού δηλώσουμε τον επιθυμητό interpreter (πρώτη γραμμή) ορίζουμε μερικές μεταβλητές που θα χρειαστούν στην πορεία. Οι πρώτες δύο αξιοποιούνται στο τέλος του script, για την ασφαλή μεταφορά του συνθηματικού στο μηχάνημα από το

οποίο πραγματοποιούμε τις συνδέσεις. Παρατηρήστε ότι για τη διεύθυνση αυτού του μηχανήματος διαβάζουμε μια μεταβλητή του περιβάλλοντος. Πρόκειται για την $SSH_CLIENT, η οποία περιλαμβάνει μεταξύ άλλων και τη διεύθυνση από την οποία έχουμε συνδεθεί! Η μεταβλητή count αποτελεί έναν μετρητή που χρησιμοποιείται στον κύριο βρόχο του script, ενώ η sequence θα φιλοξενήσει το νέο συνθηματικό.

```
while [ $count -lt 5 ];
do
      port=$(( $RANDOM % 16000 ))
      port=$(( $port + 16000 ))
      sequence="${sequence},${port}"
      count=$(( $count + 1 ))
done
```

Ο κύριος βρόχος του script! Σύμφωνα με τον έλεγχο που πραγματοποιείται στην πρώτη γραμμή, το περιεχόμενό του βρόχου εκτελείται πέντε φορές. Ο βρόχος σχηματίζει ένα string με την αλληλουχία των δικτυακών θυρών, που θ' αποτελεί το νέο συνθηματικό χτύπημα. Σε κάθε επανάληψη του βρόχου λαμβάνουμε μια τυχαία τιμή από την ειδική μεταβλητή $RANDOM του BASH. Προσέξτε ότι ταυτόχρονα πραγμα-


pvar@crunchbang: ~
GNU nano 2.2.6 File: new_sequence.sh Modified

```
#!/bin/bash

remote_user="pvar"
remote_syst=`echo $SSH_CLIENT | awk {'print $1'}`

count=0
sequence=""

while [ $count -lt 5 ];
do
        port=$(( $RANDOM % 16000 ))
        port=$(( $port + 16000 ))

        sequence="${sequence},${port}"

        count=$(( $count + 1 ))
        first=0
done

echo $sequence > ~/.sequence

scp ~/.sequence ${remote_user}@${remote_syst}:~/

echo; sudo /etc/init.d/knockd restart
```

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

Το κεντρικό εξάρτημα του μηχανισμού που παρουσιάζουμε: Το script που μετά από κάθε σύνδεση δημιουργεί μια αλληλουχία χτυπημάτων, φροντίζει να την αντιγράψει στο τοπικό μας σύστημα και, φυσικά, ενημερώνει το knockd για την αλλαγή.



Για δική μας ευκολία, δημιουργήσαμε ένα script που αξιοποιεί το νέο κάθε φορά συνθηματικό, για να ξεκλειδώσει τον server και να πραγματοποιήσει μια σύνδεση SSH.

τοποιούμε την πράξη modulo[1] με τον αριθμό 16000, ώστε να πάρουμε μια τυχαία τιμή στο διάστημα [0, 16000]. Σε αυτή την τιμή προσθέτουμε το 16000 κι έτσι η τυχαία τιμή ανήκει πλέον στο διάστημα [16000, 32000]. Με αυτόν τον τρόπο εξασφαλίζουμε ότι οι θύρες που απαρτίζουν το συνθηματικό χτύπημα βρίσκονται σε ασφαλή περιοχή: Μακριά από θύρες που χρησιμοποιούνται παραδοσιακά για γνωστές υπηρεσίες και οι οποίες, ως εκ τούτου, είναι πιθανό να δέχονται χτυπήματα (αιτήματα) ούτως ή άλλως.

```
echo $sequence > ~/.sequence
scp ~/.sequence ${remote_user}@${remote_syst}:~/
echo; sudo /etc/init.d/knockd restart
```

Εφόσον έχει δημιουργηθεί το νέο συνθηματικό, αρκεί να το αποθηκεύσουμε στην κατάλληλη θέση (~/.sequence), όπως επίσης και να το στείλουμε στο μηχάνημα από το οποίο έχουμε συνδεθεί. Όπως έχουμε πει ήδη, η αποστολή πραγματοποιείται με το secure copy, που δημιουργεί αυτόματα μία κρυπτογραφημένη σύνδεση με τη βοήθεια του SSH. Σημειώστε ότι σε αυτό το στάδιο της εκτέλεσης του script, θα χρειαστεί να δώσουμε το απαιτούμενο συνθηματικό για τη σύνδεση του scp. Το script ολοκληρώνεται με την επανεκκίνηση του knockd. Κάτι τέτοιο είναι επιβεβλημένο, καθώς το αρχείο με το συνθηματικό ανανεώθηκε και ο knockd πρέπει να ενημερωθεί! Για την επανεκκίνηση της υπηρεσίας απαιτούνται αυξημένα δικαιώματα και γι' αυτό ακριβώς επιστρατεύουμε το sudo. Αυτό συνεπάγεται ότι η εκτέλεση του script

1. Επιστέφει το υπόλοιπο της ακέραιης διαίρεσης δύο ακεραίων αριθμών.

θα διακόπτεται *και* σε αυτό το σημείο, για να πληκτρολογούμε το συνθηματικό μας. Σ' αυτό το σημείο οφείλουμε να παραδεχτούμε ότι οι διακοπές του script για την πληκτρολόγηση συνθηματικών θα καταντήσουν γρήγορα πολύ εκνευριστικές. Στην πορεία θα δούμε πώς μπορούμε να απαλλαγούμε από την εισαγωγή του συνθηματικού.

Έχοντας ετοιμάσει το script, απομένει να το τοποθετήσουμε στη σωστή θέση (~/bin) και με το σωστό όνομα (new_sequence.sh), ώστε να εκτελείται αυτόματα από το ~/.bash_profile. Τέλος, αφού φροντίσουμε να το καταστήσουμε εκτελέσιμο (chmod u+x new_sequence.sh) πρέπει και να το τρέξουμε. Αυτό είναι αναγκαίο, διότι μέχρι στιγμής δεν είχαμε ορίσει κανένα συνθηματικό για το knockd.

Σουσάμι άνοιξε...

Τώρα που το port knocking είναι πλήρως αυτοματοποιημένο στον server, καλό θα ήταν να το αυτοματοποιήσουμε και στο μηχάνημα από το οποίο πραγματοποιούμε τις συνδέσεις. Εξαιτίας της αντιγραφής που πραγματοποιεί το scp, το νέο συνθηματικό τοποθετείται στη θέση ~/.sequence. Για λόγους ευκολίας, λοιπόν, μπορούμε να φτιάξουμε ένα μικρό script (http://bit.ly/dh032unlock) που θα διαβάζει το συνθηματικό, θα πραγματοποιεί τα σχετικά χτυπήματα στον server κι ακολούθως θα δημιουργεί μια σύνδεση SSH. Ο κώδικας του script είναι αρκετά απλός και σίγουρα κατανοητός:

```
#!/bin/bash
server="192.168.1.105"
username="pvar"
while read line;
do
    knock $server $line
done < ~/.sequence
sleep 1
ssh ${username}@${server}
```

...και τώρα κλείσε!

Αφού φτιάξαμε ένα σκριπτάκι για το άνοιγμα της θύρας 22, θα μπορούσαμε να φτιάξουμε κι ένα script για το κλείσιμό της. Όπως θα θυμόσαστε, γι' αυτή τη λειτουργία έχουμε ορίσει ένα σταθερό συνθηματικό χτύπημα. Ωστόσο, δεν υπάρχει κανένας λόγος να το μάθουμε απέξω. Αρκεί να γράψουμε ένα script της μίας γραμμής:

```
#!/bin/bash
server="192.168.1.105"
knock $server 6009,7008,8007,9006
```

Τέλος ή αρχή;

Τα σκριπτάκια μας μπήκαν στη θέση τους και λειτουργούν άψογα ή, τέλος πάντων, κάνουν αυτό που περιμένουμε. Μήπως όμως ξεχνάμε κάτι; Μπορεί ο μηχανισμός με τα scripts να δουλεύει, αλλά αυτό δεν σημαίνει ότι είναι τέλειος. Κατ' αρχάς, για να λειτουργήσει το όλο σύστημα πρέπει να πραγματοποιούμε συνδέσεις στον

server *πάντα* από τον ίδιο υπολογιστή. Σε διαφορετική περίπτωση, το μηχάνημα που έχουμε μπροστά μας δεν θα διαθέτει το ανανεωμένο συνθηματικό. Η αλήθεια είναι ότι γι' αυτό το πρόβλημα δεν μπορούμε να κάνουμε πολλά. Υπάρχει όμως και μια άλλη ατέλεια, η οποία ευτυχώς μπορεί να διορθωθεί σε μεγάλο βαθμό. Σε κάθε σύνδεση προς το server πρέπει να πληκτρολογούμε τρία συνθηματικά: Ένα για να συνδεθούμε, ένα για να πραγματοποιηθεί η αντιγραφή του νέου συνθηματικού κι ένα ακόμα για την επανεκκίνηση του knockd. Προφανώς, αυτό είναι τουλάχιστον κουραστικό. Μια καλή λύση θα ήταν να δημιουργήσουμε ένα ζεύγος ιδιωτικού-δημόσιου κλειδιού σε κάθε μηχάνημα και, ακολούθως, να εγκαταστήσουμε το δημόσιο κλειδί του ενός συστήματος στο άλλο. Με αυτόν τον τρόπο, οι συνδέσεις SSH από το ένα μηχάνημα προς άλλο, θα πραγματοποιούνται χωρίς την ανάγκη εισαγωγής συνθηματικών. Ας δούμε τη διαδικασία ξεκινώντας από τον server:

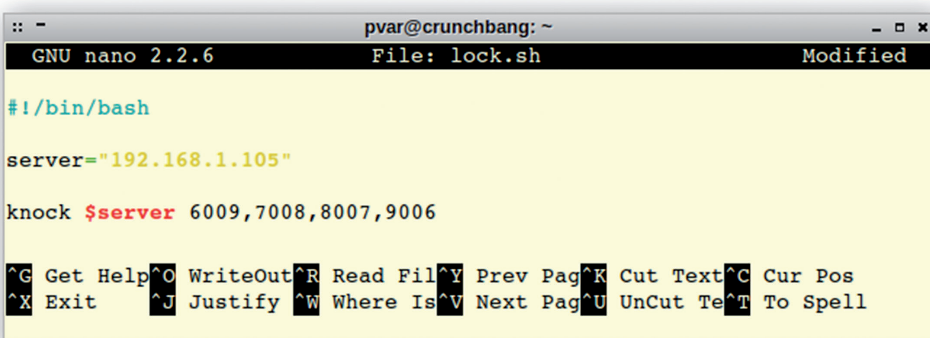
```
ssh-keygen -q -f ~/.ssh/id_rsa -t rsa
```

Στις ερωτήσεις που θα μας απευθύνει το σύστημα (για τη δημιουργία passphrase) πρέπει να πατήσουμε [Enter] και να μην εισάγουμε τίποτα. Διαφορετικά, σε κάθε σύνδεση SSH θα πρέπει να δίνουμε το passphrase! Αφού δημιουργηθεί το ζεύγος, αρκεί να αντιγράψουμε το δημόσιο κλειδί στο τοπικό μηχάνημα:

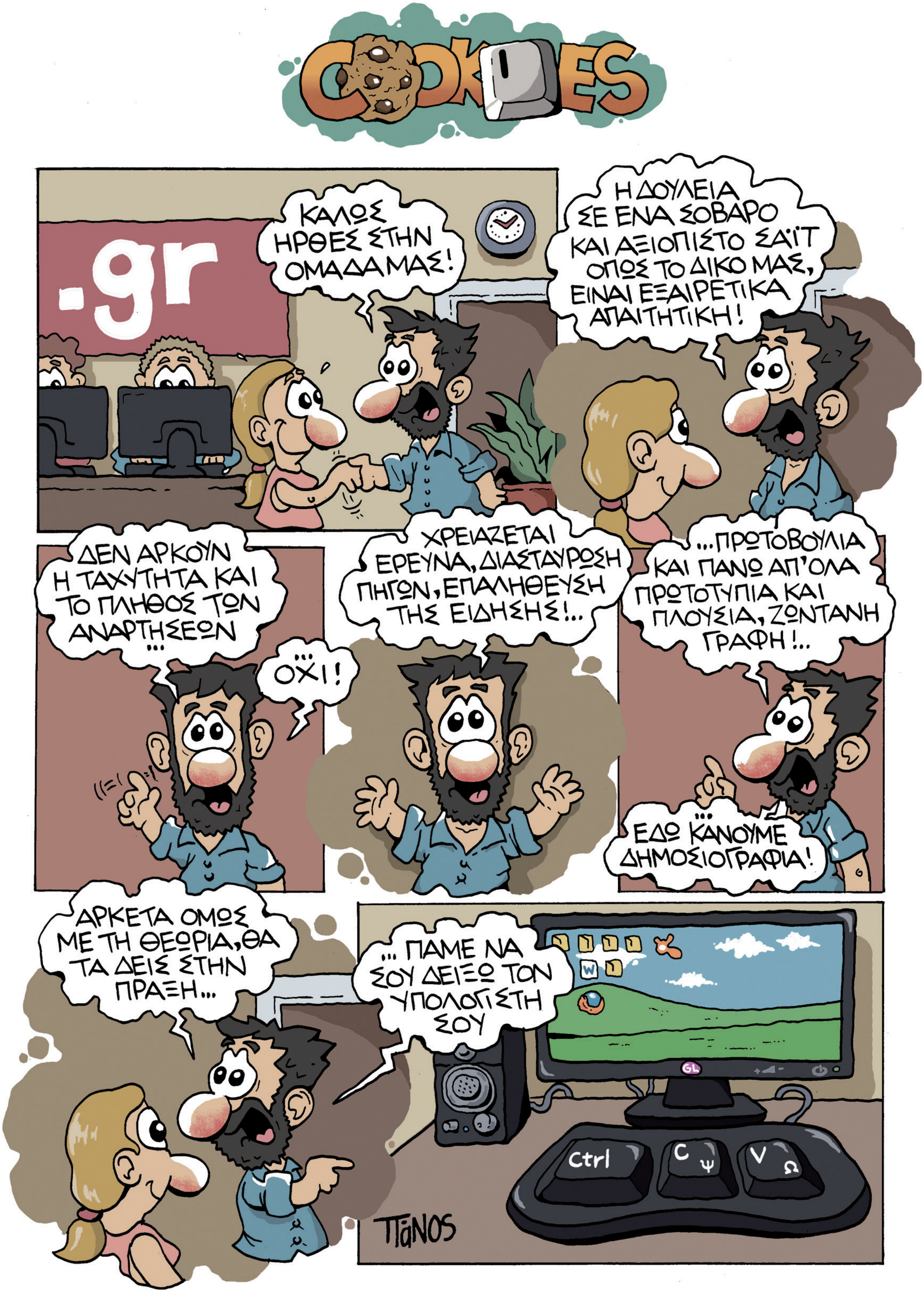
```
scp ~/.ssh/id_rsa pvar@my_home_pc:~/.ssh/authorized_keys
```

Τέλος, αρκεί να δημιουργήσουμε ένα ζεύγος κλειδιών και στο τοπικό σύστημα, όπως επίσης και να αντιγράψουμε το δημόσιο κλειδί στον αντίστοιχο κατάλογο του server. Σημειώστε ότι τα ζεύγη κλειδιών δημιουργήθηκαν από το χρήστη pvar. Επιπρόσθετα, τα δημόσια κλειδιά τοποθετήθηκαν στον κατάλογο .ssh, μέσα στο home directory του pvar. Αυτό σημαίνει ότι ο αυτοματισμός που θέλουμε να πετύχουμε (σύνδεση SSH χωρίς την πληκτρολόγηση συνθηματικού) θα ισχύει μόνο για συνδέσεις με το λογαριασμό του pvar.

Με το μηχανισμό που εξετάσαμε σε αυτό το άρθρο, οι συνδέσεις μας σε κάποιον server μπορούν να γίνουν λίγο πιο ασφαλείς και σίγουρα πιο άνετες! Ωστόσο, πιστεύουμε ότι υπάρχουν πολλά περιθώρια βελτιώσεων, τόσο στον τομέα της ασφάλειας, όσο και σε αυτόν της ευκολίας. Εσείς τι θα βελτιώνατε;



Για την ευκολία μας (είμαστε καλομαθημένοι) δημιουργήσαμε ένα μικροσκοπικό script, το οποίο πραγματοποιεί την κατάλληλη αλληλουχία χτυπημάτων στο server ώστε να κλείσει το port 22.


COOKIES
.gr
ΚΑΛΩΣ ΗΡΘΕΣ ΣΤΗΝ ΟΜΑΔΑ ΜΑΣ!
Η ΔΟΥΛΕΙΑ ΣΕ ΕΝΑ ΣΟΒΑΡΟ ΚΑΙ ΑΞΙΟΠΙΣΤΟ ΣΑΪΤ ΟΠΩΣ ΤΟ ΔΙΚΟ ΜΑΣ, ΕΙΝΑΙ ΕΞΑΙΡΕΤΙΚΑ ΑΠΑΙΤΗΤΙΚΗ!
ΔΕΝ ΑΡΚΟΥΝ Η ΤΑΧΥΤΗΤΑ ΚΑΙ ΤΟ ΠΛΗΘΟΣ ΤΩΝ ΑΝΑΡΤΗΣΕΩΝ...
...ΟΧΙ!
ΧΡΕΙΑΖΕΤΑΙ ΕΡΕΥΝΑ, ΔΙΑΣΤΑΥΡΩΣΗ ΠΗΓΩΝ, ΕΠΑΛΗΘΕΥΣΗ ΤΗΣ ΕΙΔΗΣΗΣ!...
...ΠΡΩΤΟΒΟΥΛΙΑ ΚΑΙ ΠΑΝΩ ΑΠ'ΟΛΑ ΠΡΩΤΟΤΥΠΙΑ ΚΑΙ ΠΛΟΥΣΙΑ, ΖΩΝΤΑΝΗ ΓΡΑΦΗ!...
...ΕΔΩ ΚΑΝΟΥΜΕ ΔΗΜΟΣΙΟΓΡΑΦΙΑ!
ΑΡΚΕΤΑ ΟΜΩΣ ΜΕ ΤΗ ΘΕΩΡΙΑ, ΘΑ ΤΑ ΔΕΙΣ ΣΤΗΝ ΠΡΑΞΗ...
...ΠΑΜΕ ΝΑ ΣΟΥ ΔΕΙΞΩ ΤΟΝ ΥΠΟΛΟΓΙΣΤΗ ΣΟΥ
Ctrl
C Ψ
V Ω